Robustness in timed automata : analysis, synthesis, implementation

Ocan Sankur

EDSP
École Doctorale
Sciences Pratiques

ENSC-2013/445

ENS
CACHAN

THÈSE DE DOCTORAT
DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

présentée par

Ocan SANKUR

pour obtenir le grade de

DOCTEUR DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Domaine :

Informatique

Sujet de la thèse :

Robustness in Timed Automata: Analysis, Synthesis, Implementation

Thèse présentée et soutenue à Cachan le 24 Mai 2013 devant le jury composé de :

Rajeev Alur	Professeur, University of Pennsylvania	Rapporteur
Eugène Asarin	Professeur, Université Paris-Diderot	Rapporteur
Patricia Bouyer	Directrice de recherche, CNRS & ENS Cachan	Directrice de thèse
Krishnendu Chatterjee	Assistant Professor, IST Austria	Examinateur
Claude Jard	Professeur, Université de Nantes	Examinateur
Kim G. Larsen	Professeur, Aalborg University	Examinateur
Nicolas Markey	Chargé de recherche, CNRS & ENS Cachan	Directeur de thèse
Jean-François Raskin	Professeur, Université Libre de Bruxelles	Rapporteur

ENS
CACHAN

LSV – ENS Cachan
61 avenue du président Wilson – F-94230 Cachan

Acknowledgement

I am grateful to Patricia and Nicolas who introduced me to timed automata and robustness problems. Throughout my master's and PhD, we explored many aspects of timed automata; they have been supportive, and always available for discussions and advice. They encouraged me in pursuing my curiosities, and it has been a great experience working with them. Some of the figures created with the TikZ package in this thesis, especially the nicer ones, were made by Nicolas.

I would like to thank Rajeev Alur, Eugène Asarin, and Jean-François Raskin for accepting to review this thesis. Many thanks to Krishnendu Chatterjee, Claude Jard, and Kim G. Larsen for accepting to be in my jury.

My parents deserve special thanks, as they always privileged (the education of) their children, and asked nothing in return. I promise to pursue the work on the Turkish scientific terminology. As a modest proof, I provide an abstract of my thesis in Turkish.

Special thanks go to all coffee growers world-wide; I believe I have drunk about ten thousand cups of coffee since I started to work on computer programming and science.

Thanks to Romain for many interesting discussions, and to Stefan as well for research adventures on a different topic.

Thanks to Benjamin and Aiswarya for organizing *goûters des doctorants*, which allowed us to regularly meet and discuss different topics, while enjoying home-made pastry.

And of course, thanks to my beloved Agathe for her continuous support.

Résumé

Les automates temporisés sont un formalisme qui permet de modéliser, vérifier, et synthétiser des systèmes temps-réels. Ils sont dotés d'une sémantique abstraite et mathématique, qui permet de formaliser et résoudre plusieurs problèmes de vérification et de synthèse. Cependant, les automates temporisés sont utilisés pour concevoir des modèles, plutôt que décrire des systèmes temps-réels entiers. Ainsi, une fois la phase de conception terminée, il reste à déterminer si les comportements du modèle correspondent à ceux d'un vrai système. Une étape importante de l'implémentation consiste à s'assurer de la robustesse du système. On considère une notion de robustesse sur les automates temporisés qui exige que les comportements soient préservés quand le modèle est sujet à des perturbations bornées. Dans cette thèse, plusieurs approches sont étudiées : Dans l'analyse de robustesse, on se demande si un automate temporisés donné préserve ses comportements sous divers types de perturbations, et on cherche à calculer un majorant sur les perturbations tolérées. La synthèse robuste s'intéresse au calcul d'une loi de contrôle (ou une *stratégie*) qui guide le système, et tolère des perturbations d'une magnitude calculable. Enfin, dans l'implémentation robuste, on s'intéresse à transformer automatiquement un modèle donné pour le rendre robuste, tout en préservant ses comportements.

Plusieurs modèles de perturbations sont considérés : erreurs de mesure de temps (élargissement de gardes), élimination des comportements limites (contraction de gardes), et la restriction du domaine du temps aux valeurs discrètes. On formalise également les problèmes de synthèse robuste comme des jeux entre le contrôleur et un environnement qui perturbe systèmatiquement tout délai choisi par une quantité bornée. Ces problèmes sont étudiés pour les automates temporisés, ainsi que leurs extensions; les jeux temporisés, et les automates et jeux temporisés pondérés.

Plusieurs algorithmes d'analyse de robustesse parametrée contre l'élargissement de gardes et la contraction de gardes sont proposés. Deux variantes de la sémantique de jeu pour le problème de synthèse robuste sont également étudiées pour les automates temporisés et leurs extensions. Un logiciel d'analyse de robustesse contre la contraction de gardes, ainsi que des résultats expérimentaux sont présentés. Le problème de l'implémentation robuste est étudié dans deux contextes différents. Tous les algorithmes calculent également un majorant sur les perturbations que le modèle donné est capable de tolérer.

Özet

Zamanlı otomatlar, gerçek zamanlı sistemlerin modellenmesi, doğrulanması, ve sentezlenmesine olanak sağlayan bir formalizmdir. Bu modelin soyut ve matematiksel semantiği, bu sistemler üzerinde birçok problemin biçimsel olarak tanımlanıp bilgisayarla çözülmesini olanaklı kılar. Öte yandan, zamanlı otomatlar sistemin kendisinden ziyade soyut bir modelini ifade etmek için kullanılır. Dolayısıyla, doğrulama süreci bittiğinde, soyut modelin gerçekleştiriminin ne derece model ile aynı davranışlara sahip olduğunun saptanması gerekir. Bir modelin gerçekleştirilmesi sürecinde önemli bir adım dayanıklılığının sağlanmasıdır. Gerçek zamanlı sistemler bağlamında dayanıklılık, sistemin zamanlaması gürültüye maruz kaldığında, sistemin davranışının korunmasını gerektirir. Dayanıklılığın incelenmesinde birçok yaklaşım ele alınmıştır: Dayanıklılık çözümlemesinde amaç, verilen bir sistemin gürültü altında davranışlar kümesinin değişip değişmediğinin saptanması, ve mümkünse sistemin dayandığı ençok gürültü miktarının hesaplanmasıdır. Dayanıklı sentez, verilen bir sistemi yöneten ve gürültüye dayanıklı bir kontrol yasası (veya strateji) hesaplanmasıyla ilgilenir. Dayanıklı gerçekleştirimde ise amaç, verilen bir zamanlı sistemi otomatik olarak, davranışlarını değiştirmeden gürültüye dayanıklı hale getirmektir.

Bu tezde birçok parametreli gürültü modeli ele alınmıştır. Bu modeller arasında, zaman ölçümüne belirsizlik eklenmesi (önkoşul genişlemesi), zaman koşullarının sınırında bulunan davranışların göz ardı edilmesi (önkoşul daraltması), zaman doğrusunun örneklemlenerek kısıtlanması vardır. Ayrıca, dayanıklı sentez problemi, kontrol yasası ile, yasanın seçtiği girdileri sistematik olarak değiştiren bir ortam arasında oynanan bir oyun olarak modellenip incelenmiştir. Bu gürültü modelleri zamanlı otomatlar, ve bu formalizmin uzantıları olan zamanlı oyunlar, ve ağırlıklı zamanlı otomat ve oyunlara uygulanmıştır.

Bu çalışmada sunulan sonuçlar, önkoşul genişlemesi ve daraltması modelleri için parametrik dayanıklılık çözümlemesi algoritmaları, dayanıklı sentez probleminin iki çeşit oyun semantiğinde, zamanlı otomatlar, oyunlar, ve ağırlıklı uzantıları için çözümleridir. Ayrıca, iki değişik bağlamda dayanıklı gerçekleştirim sorunu ele alınmıştır. İşlenen tüm algoritmalar verilen zamanlı otomatın dayandığı gürültü miktarının bir üstsınırını hesaplar. Önerilen önkoşul daraltmasına karşı dayanıklılık çözümlemesini gerçekleştiren bir yazılım hazırlanmış, bilimsel yazında ele alınmış olan birçok model üzerinde sınanmıştır.

Abstract

Timed automata are a formalism to model, verify, and synthesize real-time systems. They have the advantage of having an abstract mathematical semantics, which allow formalizing and solving several verification and synthesis problems. However, timed automata are intended to design models, rather than completely describe real systems. Therefore, once the design phase is over, it remains to check whether the behavior of an actual implementation corresponds to that of the timed automaton model. An important step before implementing a system design is ensuring its robustness. This thesis considers a notion of robustness that asks whether the behavior of a given timed automaton is preserved, or can be made so, when it is subject to small perturbations. Several approaches are considered: Robustness analysis seeks to decide whether a given timed automaton tolerates perturbations, and in that case to compute the (maximum) amount of tolerated perturbations. In robust synthesis, a given system needs to be controlled by a law (or *strategy*) which tolerates perturbations upto some computable amount. In robust implementation, one seeks to automatically transform a given timed automaton model so that it tolerates perturbations by construction.

Several perturbation models are considered, ranging from introducing error in time measures (guard enlargement), forbidding behaviors that are too close to boundaries (guard shrinking), and restricting the time domain to a discrete sampling. We also formalize robust synthesis problems as games, where the control law *plays* against the environment which can systematically perturb the chosen moves, by some bounded amount. These problems are studied on timed automata and their variants, namely, timed games, and weighted timed automata and games.

Algorithms for the parameterized robustness analysis against guard enlargements, and guard shrinkings are presented. The robust synthesis problem is studied for two variants of the game semantics, for timed automata, games, and their weighted extensions. A software tool for robustness analysis against guard shrinkings is presented, and experimental results are discussed. The robust implementation problem is also studied in two different settings. In all algorithms, an upper bound on perturbations that the given timed automaton tolerates can be computed.

Contents

Chapter 1

Introduction

1.1 Real-time systems

Digital systems have become a part of our lives: personal computers, smartphones, even espresso machines, but also parts of cars, trains, and airplanes. In fact, thanks to the wide availability of microprocessors, many physical systems are today controlled by computers. For instance, the automotive and the aerospace industries have long been turned to X-by-wire systems, where the control mechanism is ensured by electronic components. Among these systems, *real-time systems* are of particular interest. These are heterogeneous systems made of physical machines and software parts, and are moreover subject to real-time constraints such as response times and operational deadlines. Thus, designing such systems not only involves developing correct software in the classical sense, but also requires ensuring task execution times, response time guarantees, and other quantitative constraints such as low energy consumption. As real-time systems have gained in complexity, the correctness of critical systems (*e.g.* airplanes, nuclear plants), and consequently the safety of people involved in these systems have become an issue. In complex real-time systems, small design errors can stay unnoticed during the development and testing phase and have serious consequences. While some of these errors are implementation errors that are introduced during development, some others are design errors that could be corrected before manufacturing even begins. Some publicly known examples to such errors is the Toyota Prius brake system software error[1], which caused a reaction delay of the braking system of almost one second, and the priority inversion bug in the NASA Mars Pathfinder[2], which required restarting the system, thus reducing precious cruise time.

Why are some serious errors not detected? One reason is perhaps the difficulty of applying techniques from software engineering to real-time systems. For instance, the classical testing and debugging of real-time systems is often difficult, since reproducing a physical state of the environment and of the system is often impossible. Also, one cannot simply stop time while debugging and analyze the system. Another source of complexity is due to the distributed nature of some real-time systems, which is already a known issue for concurrent software systems. Most importantly, usual techniques for software development do not take into account *non-functional properties* such as timing constraints in the executions of the system and physical quantities on which the behavior of the system may depend. Classical control theory does not provide a solution neither since it is not well suited for systems with a software control structure. A comparison between developing systems with classical control theory and developing embedded software systems is given in [CM05]. In fact, real-time systems require specialized techniques, depending on the application area; see *e.g.* [Kop11]. Finding design methods and techniques to ensure the correctness of various kinds of real-time systems is largely an open research problem today. In [HS06], the authors identify the design of embedded systems as an important challenge, and suggest research directions to develop theories and tools for the development of reliable embedded systems.

[1] http://www.reuters.com/article/2010/02/09/toyota-recall-announcement-idAFTKG00664220100209
[2] http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/Authoritative_Account.html

1.2 Formal Verification and Timed Automata

Formal verification consists in proving or disproving the correctness of the design of a system described in a mathematical formalism. Several techniques have been developped for formal verification. Among these, *model-checking* has an important place. It consists in an exhaustive exploration of the mathematical model of a system, in order to prove or disprove that it satisfies a given property. Model-checking has first been applied to verify finite-state models against properties expressed in temporal logic; see [GV08] for the first applications of model-checking and the history of its development. Richer formalisms have then been considered in order to apply model-checking on infinite-state systems that have unbounded data, communication, probabilities, and time.

In this thesis, we concentrate on model-checking *timed automata* [AD90, AD94], an extension of finite automata with analog clock variables. This is an abstract and convenient formalism to express several kinds of systems with timing constraints. Several model-checking tools are available for timed automata such as Uppaal [BDL$^+$06], Kronos [BDM$^+$98], RED [Wan06], Rabbit [BLN03]. The timed automata technology has been used to prove the correctness of communication protocols [DY00], clock synchronization algorithms [RNPH05], audio/video protocols [HSLL97], and tool support is available specifically for scheduling analysis of real-time tasks, see *e.g.* the Times Tool [AFM$^+$02].

We will explain timed automata on an example. We consider a simple model of a computer mouse controller whose aim is to distinguish between simple and double clicks. We would like any click to be interpreted as a double click whenever it is preceeded by a simple click in less than 100 milliseconds. The timed automaton is given in Fig. 1.1. The underlying finite automaton is simple: it has two locations, and three transitions labelled by click. Notice that the system does not make much sense if we do not take into account time delays between actions: seen as a finite automaton, the system can generate any sequence of clicks, while it can return to location double only after going through location simple. Timing constraints are ensured by the clock x. In timed automata, clocks always grow at a constant rate, and cannot be stopped; they can only be *reset* (to 0) during transitions. Here, the clock x is reset during clicks. Furthermore, the transition from simple to double is *guarded* by $x < 100$, which means that a double click is realizable only if $x < 100$. A simple click is realizable after a previous click if $x \geq 100$, that is, at least 100 milliseconds has elapsed since the latest click.

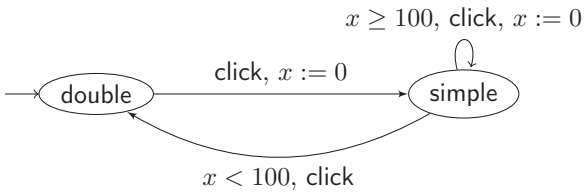


Figure 1.1: A timed automaton specifying a simple mouse controller detecting simple and double clicks. The clock x is reset when a simple click is realized. After that, any click event received in less than 100 time units gives rise to a double click. Otherwise it is another simple click.

Model-checking algorithms for timed automata allow one to exhaustively verify whether all runs satisfy a property. The problem was shown to be decidable; for instance, PSPACE-complete for location-based Büchi properties [AD94], and for timed properties expressed in the *timed computation-tree logic* [ACD93]. For example, on the timed automaton of Fig. 1.1, one could check whether

two consecutive visits to double are possible (it is not), or whether two consecutive visits to simple separated by more than 100 milliseconds are possible (it is).

1.3 Robustness

In this thesis, we are interested in extending model-checking algorithms to take into account *robustness* constraints, that is, not only checking the correctness of the behaviors in the standard semantics, but also checking for vulnerabilities of a given timed automaton to small perturbations in timings. Robustness is a crucial property of hard real-time systems, requiring them to resist to errors or failures, such as unexpected input; the behavior of a good real-time system should not change drastically if a small error occurs. In control theory, robustness is expressed in terms of continuity: The response of a system should be a continuous function of its input, thus small variations in the input should cause small variations in the output. It is difficult to define such a continuity notion in real-time systems taking into account errors in discrete behavior, since the correctness of software is rather a boolean value. Thus, there is a need to define an appropriate theory of robustness for real-time systems; this was identified as an important challenge in [Hen08, HS06].

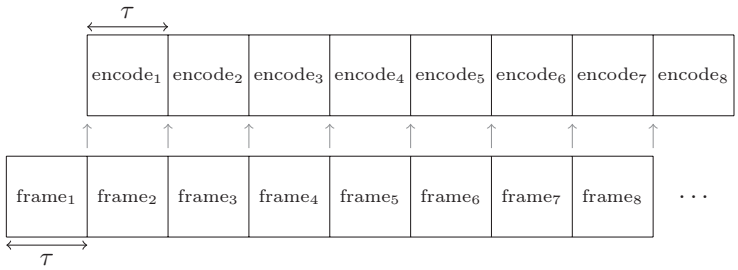


Figure 1.2: The scheduling of the tasks of a camera-recoder system. The period of the generation of the frames is the same as the encoding of the frames. The system can run for an arbitrarily long time.

Let us illustrate robustness on a simple real-time system. We consider a camera that captures a frame every τ time units, while an encoder treats a frame with the same period. We assume that the camera sends a frame to the encoder through a digital communication medium, and that the encoder has a bounded buffer which it uses to store incoming frames. The expected behavior of the system is illustrated in Fig. 1.2. We abstract away the computations and only concentrate on execution times. Thus, we are interested in the schedulability of these tasks. We would like the system to be able to treat an infinite number of tasks. This means that the encoder should not be too late treating the frames since it has only a finite buffer. Since both periods are the same, the present system is clearly capable of realizing this specification.

Now that we assessed that the "exact" behavior of the system is correct, the next step should be to check for robustness. Does the system still satisfy the specification under small perturbations? Let us consider a simple perturbation which consists in increasing or decreasing by a small amount the execution times. For instance, assume that the camera works slightly faster than expected, *e.g.* it sends frames with a period of $\tau - \epsilon$, while the encoder treats each frame in τ time units. The scheduling of the tasks is shown in Fig. 1.3. Since the encoder is now slower than the camera, it will

eventually skip frames or have a buffer overflow. The system no longer satisfies the specification (being able to run indefinitely) however small $\epsilon > 0$ may be. The system is, therefore, not robust.

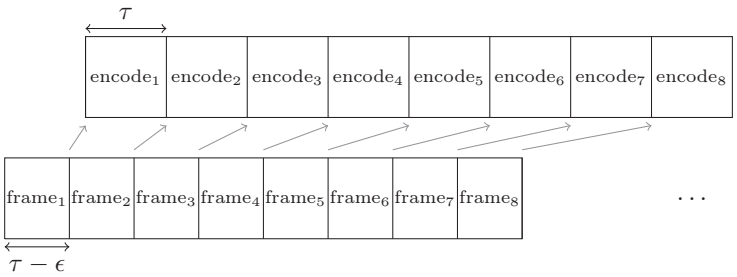


Figure 1.3: The scheduling of the tasks of a camera-recoder system of Fig. 1.2, where the execution times are slightly changed. The period of the generation of the frames, $\tau - \epsilon$, is shorter than the period of encoding. For any value of $\epsilon > 0$, the encoder will have an overflow or it will skip frames eventually.

The reason behind non-robustness phenomena is that the mathematical model is often an idealization of a physical system. For instance, in the above example, we have summarized the behavior of a complex system as a simple periodic event of duration τ. Clearly, no such physical system can be guaranteed to perform a complex task in *exactly* τ time units. Often, one can only compute or estimate lower and upper bounds on the execution times, and even that is a challenging problem [Kop11]. Nevertheless, modelling with *logical execution times* as in Fig. 1.2 is useful for a first check of whether the abstract design is correct, and has the advantage of being simple (see *e.g.* [HHK01]). When modelling at such a high level of abstraction, the system designer knows that she will have to make sure that timing errors are indeed negligible, before actually implementing the system. Similar ideas appear in synchronous programming paradigm [Ber00], where communications between components are assumed to be instantaneous.

Timed automata are a generic tool that allows modelling systems at various levels of abstractions. They allow modelling scheduling systems as the one described above, but also systems where time has a different scale: one time unit can represent nanoseconds when analyzing stabilization times for asynchronous circuits; it can represent one second in a train-gate controller, where significant events are separated by delays of higher order of magnitude. Thus, timed automata are equipped with a general-purpose idealized mathematical semantics: Time delays can be arbitrarily small and precise, actions (location changes) are instantaneous, and when there are several clocks, these grow at the same rate. This semantics has the advantage of being formal, simple and easy to analyze.

Consequently, robustness problems as in Fig. 1.3 are an issue for timed automata models. This advocates for the development of formal techniques for ensuring the robustness of timed automata. The main objectives in the study of robustness in timed automata can be summarized as follows:

1. *Robustness analysis*, where one checks whether a given timed automaton satisfies its specification when it is subject to perturbations.

2. *Robust synthesis*, where one synthesizes a controller given as a timed automaton, which can enforce a correct behavior under perturbations.

3. *Robust implementation*, where one seeks to (automatically) transform a timed automaton into an equivalent one whose behavior respects the specification under perturbations.

4. *Language theory,* where the goal is to understand to what extent the idealistic semantics of timed automata correspond to that of real-time systems.

We will address all four questions in this thesis. Let us first summarize previous work on robustness in timed automata.

1.4 Previous Work on Robustness

We overview several perturbation models considered in the timed automata literature to study robustness. Because timed automata are a general-purpose formalism, several perturbation models have been considered. In fact, whether the idealistic assumptions behind the semantics are justified or not depends on a given application. For instance, if clocks model several time measures done by a single processor, then assuming they are synchronous is harmless. However, if the timed automaton represents the global behavior of a distributed system, say, with several processes each having an independent clock, then one should verify that the specification is satisfied when these clocks are subject to drift. Hence, the study of robustness calls for specific theories for each application area.

In [Pur00], Puri studies the semantics of closed timed automata where the clocks are subject to a drift bounded by some $\epsilon > 0$. More precisely, the semantics is modified by allowing the clocks to grow with independent rates from $[1-\epsilon, 1+\epsilon]$. The robustness question asks whether for some $\epsilon > 0$, the semantics is safe with respect to a given set of states. The timed automaton of Fig. 2.3 appears in [Pur00] as an example of non-robust timed automaton. It is shown that one can compute the set $\bigcap_{\epsilon>0} \mathsf{Reach}_\epsilon(\mathcal{A})$ for any timed automaton $\mathcal{A}$, where $\mathsf{Reach}_\epsilon(\mathcal{A})$ denotes the set of states reachable under drifts bounded by $\epsilon > 0$. Hence, one can decide whether safety can be ensured for some $\epsilon > 0$. Robustness analysis against clock drifts was studied for more general timed automata in [Dim07]. Clock drifts yield decidability in some cases: Timed language inclusion becomes decidable if the right-hand side system is a product of one-clock timed automata under clock drifts [ALM05]. It was shown that under an external clock synchronization mechanism, closed timed automata are robust against clock drifts [SFK08]. Clock drifts were considered in [Kri00] in the context of asynchronous distributed timed automata, where clocks evolve independently in each component. Components can read each other's clocks, but cannot reset them. See also [DL07] for a variant of the model. The resulting model is more expressive than timed automata; [OLS11] suggests a model that has the same expressive power as event clock automata of [AFH99]. In [ABG$^+$08], the authors concentrate on the untimed language of distributed timed automata. They distinguish the existential and the universal (untimed) languages recognized by distributed timed automata. The existential language consists in the possible behaviors of a DTA under *some* evolution of time, where the universal language is the set of behaviors that can arise under *any* evolution of time. The former is interesting for checking safety properties and has a decidable emptiness problem. The problem is similar to that of [Pur00] except that there is no upper bound on the magnitude of the drifts. The universal semantics is useful for checking positive specifications, but emptiness is undecidable.

Another perturbation model for timed automata that appears in [Pur00] is that of guard enlargement. Here, the timed automaton is syntactically modified by *relaxing* the guards. For some parameter $\delta > 0$, enlargement consists in modifying a guard $a \leq x \leq b$, to $a - \delta \leq x \leq b + \delta$. In this setting, the robustness question asks whether for some $\delta > 0$, the timed automaton is

safe with respect to a set of states. Puri states the decidability of this question. These ideas are revisited in [DDMR08], both for clock drifts and guard enlargement, where detailed proofs are given. It turns out that the same algorithm decides the robustness against guard enlargement and clock drifts for safety objectives. Several model-checking algorithms for timed automata were re-visited in the setting of guard enlargement. More precisely, the *parameterized robust model-checking* asks whether given a timed automaton $\mathcal{A}$, there exists $\delta > 0$ such that $\mathcal{A}_\delta$, the timed automaton $\mathcal{A}$ whose all guards are enlarged by δ, satisfies a given property. For safety properties, [DDMR08] shows PSPACE-completeness, which is the complexity of the problem in the exact setting. Richer specifications were also studied: Parameterized robust model-checking against co-Büchi properties (including LTL) is PSPACE-complete [BMR06, BMS11]. For a fragment of the metric temporal logic, called coFlat-MTL, the problem is EXPSPACE-complete [BMR08] (this logic was identified earlier in [BMOW07] as a fragment of the logic MTL of [Koy90] for which model-checking is EXPSPACE-complete). Notice that the complexity of all parameterized robust model-checking algorithms is the same of the corresponding standard model-checking algorithm on timed automata. Symbolic algorithms were also studied for the subclass of timed automata without nested cycles in [DK06, JR11]. The guard enlargement approach was recently applied to time Petri nets [AHJR12].

A different line of works considers the semantics of timed automata under a sampling of time with unknown (parameterized) sampling period. More precisely, the semantics is obtained by replacing the continuous time by the multiples of a sampling period. In this setting, the emphasis is on the loss of behavior, since restricting the time domain cannot add new behaviors. Examples of timed automata whose behaviors are disabled under any sampling period appear in [CHR02]. The authors show the undecidability of the parameterized reachability problem, that is, deciding whether for some sampling period, some state is reachable, under the assumption that an action must be taken at each sampling point. Decidability is reported in [KP05] when this condition is relaxed, allowing time to elapse until any sampling point. Untimed language equivalence for some sampling period is also decidable [AKY10].

The above works concentrate on developing specific algorithms for the modified semantics of timed automata and games, in order to synthesize the parameters ϵ or δ. A different approach consists in incorporating these perturbations for some fixed values of ϵ or δ, by encoding as timed automata. This is easy to do for enlargement since, for a fixed parameter, it consists in syntactically modifying the timed automaton. One can also encode clock drifts, and synchronization delays as suggested in [AT05]. The advantage is that one can rely on existing tools to verify the resulting system. On the other hand, such an encoding generally requires using several clocks, and increases the size of the state space.

In [ACS10], robustness against changes in task execution times was investigated in real-time applications. In fact, a known phenomenon in real-time systems is that reducing execution times, for instance, by replacing the hardware by a faster one, does not always preserve correctness since the behavior of the system may depend on expected execution times. The authors describe a methodology to check, using simulation, whether reducing task execution times yield any errors.

Similar notions of robustness were considered in timed games. In [CHP11], the authors consider two-player timed games, where Player 2 can perturb the delays chosen by Player 1 by a bounded amount δ. In order to win, Player 1 is required to have a strategy that is valid under any move of Player 2, ensures a parity condition while not blocking time. The problem is studied for a fixed δ, by encoding the semantics as a regular timed game. They also give algorithms to decide the winner in this model where Player 1 is only required to play positive delays, and is subject to perturbations. This encoding in timed games were applied to robustness in interface theories in [LLTW11].

Some attempts have been made to study the semantics of timed automata models when these are implemented on physical hardware. In [DDR05a], a simple micro-processor model with a discrete clock, reaction times, and imprecisions was studied. The resulting semantics, called *program semantics*, is very detailed and is somewhat difficult to work with. The authors also give a simpler semantics, similar to the enlargement mentioned above, which over-approximates the program semantics. This gives precise motivation for the study of the enlargement in timed automata, since robustness against guard enlargement implies that the semantics of the implemented system is correct. Some case studies using this approach are reported in [DDR05b]. In [BKW12], the implementation problem of timed automata specifications is studied for commercial microcontrollers, with application on PIC microchips. In [Die01], the author defines a semantics based on timed automata for programmable logic controllers, but the resulting model is less expressive than timed automata. A related line of work is interested in code generation from timed automata models. The Times tool is capable of checking schedulability of real-time tasks and generating code [AFM+02]. Code generation from a more powerful formalism using hybrid automata is considered in [AIK+03].

Another reason for studying robustness is decidability. In fact, it is believed that the undecidability of some problems in formal verification (such as, timed language inclusion [AD94]) is due to the excessive expressive power of the formalism, *e.g.* the ability of timed automata to express complex languages using too much precision. One hope is that by introducing fuzziness in the semantics, thus excluding such unrealistic behaviors, one might obtain decidability. Several researchers argue in this direction. In [AB01] Turing machines and other hybrid systems were studied under infinitesimal perturbations. The results indicate that the languages defined by these systems become decidable in the class of robust systems, that is, those systems whose languages are unchanged under infinitesimal perturbation. Similarly, reachability becomes decidable in hybrid automata where activities are subject to small drifts, and under some conditions on the target set [Frä99]. In timed automata, an early attempt to define robustness with this idea is based on a topology defined on timed traces [GHJ97]. A run is accepted in this semantics if, and only if it has a neighborhood in which the runs of the standard semantics are dense. This allows one to introduce continuity in the semantics: any accepted run is still accepting under any small modification of the delays. Language emptiness is PSPACE-complete, which is the complexity of the problem in the standard semantics. Unfortunately, timed automata can still not be determinized under this semantics, and timed language inclusion is still undecidable [HR00, OW03a]. The authors of [OW03a] conclude that this semantics is counterproductive for obtaining decidable verification problems.

The work on *digitization*, initiated in [HMP92], seeks to simplify the verification procedure by means of restricting the domain to timed automata whose languages are stable under discretization. The paper defines a timed language to be *closed under digitization* if, roughly, all its runs executed under a periodic clock with any offset are still valid runs. *Closure under inverse digitization* requires that if all such executions yield valid runs, then the original run should be a valid one. The main result of [HMP92] is that given two timed languages $\mathcal{A}$ and $\mathcal{B}$, if $\mathcal{A}$ is closed under digitization, and $\mathcal{B}$ is closed under inverse digitization, then checking timed language inclusion is equivalent to checking timed language inclusion in discrete semantics, thus decidable. An important result derived from [HMP92] is the decidability of the inclusion of the timed language of a closed timed automaton in that of an open timed automaton, in the weakly monotonic time (that is, 0 time delays are allowed) [OW03b]. Some results of similar nature on discrete time were reported in [AMP98]. Specialized model-checkers were developed for verifying timed automata in discrete time [BMPY97, BLN03].

Table 1.1: Results on robustness analysis. Contributions are shown in shaded area.

Objective	Safety	Büchi	coFlat-MTL	Lang. Equiv.
Standard	PSPACE-c [AD94]	PSPACE-c [AD94]	EXPSPACE-c [BMOW07]	EXPSPACE-c [BGS12]
Robust	PSPACE-c [Pur00, DDMR08]	PSPACE-c [BMR06, BMS11]	EXPSPACE-c [BMR08]	EXPSPACE

1.5 Contributions of This Thesis

In this thesis, we present several results that extend the understanding of robustness in timed automata, and suggest new robustness notions. We summarize our results in four parts.

We consider the problem of *robustness analysis*, by studying the untimed language of timed automata under guard enlargement. Our result, presented in Chapter 4, is an EXPSPACEalgorithm for checking whether for some enlargement parameter, the enlarged timed automaton has the same untimed language as the original one. Table 1.1 summarizes the complexity of robustness analysis problems against guard enlargement. We then consider robustness analysis against guard *shrinkings*, which is the inverse of enlargement. In Chapter 5, we show how infinitesimal shrinkings can disable behaviors in timed automata. We give algorithms to synthesize possibly different shrinking values for each guard of a given timed automaton, so that the resulting timed automaton is non-blocking, and is able to time-abstract simulate the original one. Our algorithms are in PSPACEand EXPTIME. Chapter 6 reports on a software tool implementing these algorithms, and presents experimental results.

In the second part of this thesis, we consider the problem of *robust synthesis*. In this part, we are no more interested in verifying all behaviors of a given system, but we rather want to *robustly control* a given system, so that, even under perturbations, a desired property is ensured. We consider two perturbation game models, where a second player systematically perturbs the delays chosen by the controller. In the first one, studied in Chapter 8, the control strategy is obliged to satisfy the guards whatever perturbations are, and the goal is to synthesize strategies to ensure infinite runs satisfying a Büchi condition. This is the *conservative perturbation game* model. We prove PSPACE-completeness, and show that the problem is closely related to convergence phenomena in timed automata. In the second perturbation model, studied in Chapter 7, the control strategy only needs to suggest moves that satisfy the guards, but the perturbed moves may not satisfy these. This is the *excess perturbation game* model. The resulting semantics is different from the previous one. We prove EXPTIME-completeness for the reachability objectives on timed automata and turn-based timed games. In Chapter 9, we consider weighted timed games, for which the cost-optimal reachability problem is known to be undecidable [BBM06, BBR05a]. We prove that the undecidability "robustly" holds; in fact, the problem remains undecidable for weighted timed games in the conservative perturbation model, and it even becomes undecidable for weighted timed automata under the excess perturbation model. The latter is result is rather surprising, since it goes against the common belief that introducing perturbations can render problems more tractable. Thus, these semantics do not allow to obtain decidability for weighted timed games. On the positive side, we prove the PSPACE-completeness of the cost-optimal reachability in weighted timed *automata* under the conservative perturbation game semantics. Table 1.2 summarizes the results of this part.

In the third part, we study the *robust implementability* problem. The goal here is to transform

Table 1.2: Results on robust synthesis. TA stands for timed automata, TTG for turn-based timed games, WTA for weighted timed automata, and WTG for weighted timed games. Empty boxes indicate open problems.

Problem	Reach TA	Büchi TA	Reach TTG	OptReach WTA	OptReach WTG
Conserv.	PSPACE-c	PSPACE-c		PSPACE-c	undec.
Excess	EXPTIME-c		EXPTIME-c	undec.	undec.

given timed automata into ones that are robust by construction, yet behaviorally equivalent. We consider two approaches. In the first one, studied in Chapter 10, we show how to transform any timed automaton into one that is strongly bisimilar and whose behavior under enlargement (resp. sampling) is preserved in terms of approximate bisimilarity for any desired precision. This allows one to design using the exact semantics and then to automatically refine the system to ensure robustness with respect to guard enlargement or sampling. In a second approach, studied in Chapter 11, we use shrinking as a technique to implement timed automata on hardware. The main idea is that if all guards are shrunk, then no new behavior is present under small enough imprecisions (*i.e.* enlargement). We define a concrete implementation semantics for timed automata, similar to that of [DDR05a], including a periodic clock and communication delays, and show that the shrinkability of timed automata, in the sense of Chapter 5, implies that the implementation semantics preserves time-abstract behavior, and is non-blocking.

A fourth kind of results are more of theoretical interest, and ask whether the perturbation models considered here simplify the complexity of the verification problems on timed automata. As mentioned above, Chapter 9 presents undecidability results for the cost-optimal reachability in weighted timed games under perturbations, showing that the perturbation game semantics do not yield decidability, and can even render the problems more difficult. A second result, given at the end of Chapter 10, is a powerful one: we show that all undecidable problems on timed automata remain undecidable when restricted to the class of timed automata that are robust to guard enlargements, for any precise definition studied in the literature. Hence, perhaps counterintuitively, perturbation models based on enlargement do not decrease the expressive power of timed automata. Consequently, they do not yield the decidability of the hard verification problems.

Most of the results of this thesis have been published, with the exception of Chapters 8 and 9, which are in submission. The publications partially containing the results are following.

[BLM+11] Patricia Bouyer, Kim G. Larsen, Nicolas Markey, Ocan Sankur, and Claus Thrane. Timed automata can always be made implementable. In Joost-Pieter Katoen and Barbara König, editors, *Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR'11)*, volume 6901 of *Lecture Notes in Computer Science*, pages 76–91, Aachen, Germany, September 2011. Springer.

[BMS12] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust reachability in timed automata: A game-based approach. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP'12) – Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 128–140, Warwick, UK, July 2012. Springer.

[San11] Ocan Sankur. Untimed language preservation in timed systems. In Filip Murlak and Piotr Sankowski, editors, *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS'11)*, volume 6907 of *Lecture Notes in Computer Science*, pages 556–567, Warsaw, Poland, August 2011. Springer.

[San13] Ocan Sankur. `Shrinktech`: A tool for the robustness analysis of timed automata. In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV'13)*. To appear, 2013.

[SBM11] Ocan Sankur, Patricia Bouyer, and Nicolas Markey. Shrinking timed automata. In Supratik Chakraborty and Amit Kumar, editors, *Proceedings of the 31st Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'11)*, Leibniz International Proceedings in Informatics, pages 90–102, Mumbai, India, December 2011. Leibniz-Zentrum für Informatik.

The following ones are under submission at the time of submission of this thesis.

[BMS13] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust weighted timed automata and games. In *In submission*, 2013.

[SBMR13] Ocan Sankur, Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust controller synthesis in timed automata. In *In submission*, 2013.

The contents of the following publications are related to the results presented in this thesis. We do not include the details here in order to keep the presentation more coherent. Some of these results will however be briefly mentioned in subsequent chapters.

[BGS12] Romain Brenguier, Stefan Göller, and Ocan Sankur. A comparison of succinctly represented finite-state systems. In Maciej Koutny and Irek Ulidowski, editors, *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR'12)*, volume 7454 of *Lecture Notes in Computer Science*, pages 147–161, Newcastle, UK, September 2012. Springer.

[BMS11] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust model-checking of timed automata via pumping in channel machines. In Uli Fahrenberg and Stavros Tripakis, editors, *Proceedings of the 9th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'11)*, volume 6919 of *Lecture Notes in Computer Science*, pages 97–112, Aalborg, Denmark, September 2011. Springer.

1.6 Outline

Part I contains the preliminaries: In Chapter 2, we give formal definitions, examples, and present the perturbation models we consider in this thesis. Some important data structures used in following chapters are presented in Chapter 3. The main results are presented in three parts. Part II is on robustness analysis, and contains results on untimed language preservation under guard enlargement (Chapter 4), shrinkability (Chapter 5), and the software tool Shrinktech (Chapter 6). Part III is about robust synthesis. Chapters 7 and 8 present synthesis algorithms for, respectively, the excess perturbation game and the conservative perturbation game semantics. These semantics are applied to weighted timed games in Chapter 9. Part IV focuses on the robust implementation problem:

Chapter 10 considers approximately implementing timed automata under guard enlargement and sampling. Chapter 11 considers a more concrete model of an execution platform and studies implementation by shrinking.

Part I

Preliminaries

In this part, we give definitions and results that will be used in subsequent chapters. Chapter 2 contains formal definitions of timed automata, its variants, and their semantics. It also gives examples, and define different perturbatuion models studied in this thesis. Chapter 3 is devoted to different data structures used throughout this thesis; we also prove some technical results.

Chapter 2

Timed Automata and Their Semantics

2.1 Timed Transition Systems and Timed Game Structures

One convenient way of modelling systems is to define a state space whose elements describe all relevant information about the system at a given instant, and transitions between states, which model the evolution of the system. Formally, a system can be described by a *transition system* which is a tuple $\mathcal{T} = (S, s_0, \Sigma, \to)$ where S is an arbitrary set of *states*, $s_0 \in S$ is a distinguished *initial state*, Σ an arbitrary set of *labels*, and $\to \subseteq S \times \Sigma \times S$ is the *transition* relation. We will write transitions as $s \xrightarrow{\sigma} s'$ instead of $(s, \sigma, s') \in \to$.

Since we are interested in timed systems in this thesis, we will consider transition systems in which time is explicitly modelled. We use *timed transition systems* (TTS) as a low-level formalism to describe timed systems. Formally, a TTS is a tuple $\mathcal{T} = (S, s_0, \Sigma, \mathbb{K}, \to)$ such that $(S, s_0, \Sigma \dot\cup \mathbb{K}, \to)$ is a transition system, where $\dot\cup$ denotes disjoint union. We distinguish two types of transitions: A transition $\xrightarrow{\sigma}$ with $\sigma \in \Sigma$ is an *action*, while a transition $\xrightarrow{d}$ with $d \in \mathbb{K}$ is a *delay*. We will write $s \xrightarrow{d,\sigma} s'$ if there exists $s'' \in S$ such that $s \xrightarrow{d} s'' \xrightarrow{\sigma} s'$. Let us also denote $s \overset{\sigma}{\Rightarrow} s'$ if there exists $d \in \mathbb{K}$ such that $s \xrightarrow{d,\sigma} s'$.

A *run* of $\mathcal{T}$ is a sequence $\rho = q_1 e_1 q_2 e_2 \dots$ of states and transitions alternating between delays and actions: we have $q_i \in S$, and $q_i \xrightarrow{e_i} q_{i+1}$ for all $i \geq 1$, with $e_i \in \mathbb{K}$ for odd i, and $e_i \in \Sigma$ for even i. The run is *initialized* if $q_1 = s_0$. Let us denote by $\mathsf{Runs}(\mathcal{T})$ the set of initialized runs of $\mathcal{T}$. We denote by $\mathsf{state}_i(\rho) = q_i$ the i-th state of the run ρ, by $\mathsf{first}(\rho)$ its first state, and, if ρ is finite, $\mathsf{last}(\rho)$ denotes the last state of ρ. We also write $\mathsf{trans}_i(\rho) = e_i$. Let $|\rho|$ denote the *length* of ρ, the number of states visited by ρ. For any $0 < i < j \leq |\rho|$, we denote by $\rho_{i\dots j}$ the subrun from $\mathsf{state}_i(\rho)$ to $\mathsf{state}_j(\rho)$. The *timed trace* of ρ, denoted $\mathsf{ttrace}(\rho)$, is the sequence $(d_i, \sigma_i)_i$ of consecutive delay and actions of ρ. The *untimed trace* of ρ, denoted $\mathsf{utrace}(\rho)$, is the projection of its timed trace to the actions. We denote by $L(\mathcal{T})$ the set of untimed traces of the initialized runs of $\mathcal{T}$, called the *language* of $\mathcal{T}$. For any state $s \in S$, let $L(\mathcal{T}, s)$ denote the set of untimed traces starting at s.

A TTS $\mathcal{T} = (S, s_0, \Sigma, \mathbb{K}, \to)$ is said to be *non-blocking* if for any action $s \xrightarrow{\sigma} s'$, there exists $\sigma' \in \Sigma$ such that $s' \overset{\sigma'}{\Rightarrow} s''$. In other terms, any action can be followed, possibly after a delay, by another action. The set of reachable states, denoted $\mathsf{Reach}(\mathcal{T})$, is defined as those states that appear along initialized runs of $\mathcal{T}$.

The notions of simulation and bisimulation can be defined naturally on timed transition systems [HLY92], as follows.

Definition 2.1.1. *Given a TTS $(S, s_0, \Sigma, \mathbb{K}, \to)$, a relation $R \subseteq S \times S$ is a* timed simulation *if for any $(s, t) \in R$, and any $\alpha \in \Sigma \cup \mathbb{K}$, $s \xrightarrow{\alpha} s'$ implies that $t \xrightarrow{\alpha} t'$ for some $t' \in S$ with $(t, t') \in R$. If*

there is such a relation, we say that s is simulated by t. A timed bisimulation *is a symmetric timed simulation.*

Given two TTS $\mathcal{T}$ and $\mathcal{T}'$, we say that $\mathcal{T}$ is timed-simulated by $\mathcal{T}'$*, denoted $\mathcal{T} \sqsubseteq \mathcal{T}'$, if the initial state of $\mathcal{T}$ is simulated by the initial state of $\mathcal{T}'$ in the disjoint union of these.*

The notion of *time-abstract simulation* has been defined [LY94], in order to compare the qualitative behavior of timed automata, without requiring that the delays be matched exactly.

Definition 2.1.2. *Given a TTS $(S, s_0, \Sigma, \mathbb{K}, \to)$, a relation $R \subseteq S \times S$ is a* time-abstract simulation *if for any $(s,t) \in R$, and any $\sigma \in \Sigma$, $s \xrightarrow{\sigma} s'$ implies that $t \xrightarrow{\sigma} t'$ for some $t' \in S$ with $(t,t') \in R$, and for any $d \in \mathbb{K}$, $s \xrightarrow{d} s'$ implies that $t \xrightarrow{d'} t'$ for some $d' \in \mathbb{K}$ and $t' \in S$ with $(s',t') \in R$. If there is such a relation, we say that s is* time-abstract-simulated *by t. A* time-abstract bisimulation *is a symmetric time-abstract simulation.*

Given two TTSs $\mathcal{T}$ and $\mathcal{T}'$, we say that $\mathcal{T}$ is time-abstract-simulated by $\mathcal{T}'$*, denoted $\mathcal{T} \sqsubseteq_{t.a.} \mathcal{T}'$, if the initial state of $\mathcal{T}$ is time-abstract-simulated by the initial state of $\mathcal{T}'$ in the disjoint union of these.*

In Chapter 11, we will introduce finer notions of simulation and bisimulation, that quantify the differences between the delays of two systems.

Transition systems have generalizations to two-player games, where actions are jointly determined by both players. A *game structure* is a tuple $\mathcal{T} = (S, s_0, \Sigma_1, \Sigma_2, T_1, T_2, \mathsf{jt})$, where S is a set of *states*, $s_0 \in S$ is the *initial state*, Σ_i the set of *labels* of Player i, $T_i \subseteq S \times \Sigma_i$ is the *enabling condition* for Player i, and $\mathsf{jt} : S \times \Sigma_1 \times \Sigma_2 \to S$ the *joint transition function.* We assume that each Σ_i contains a distinguished element $\perp$ called the *empty action*, and that $\mathsf{jt}(s, \perp, \perp) = s$ for any $s \in S$. A game structure induces a transition system $(S, s_0, \Sigma_1 \times \Sigma_2, \to)$ defined by $s \xrightarrow{(\sigma_1,\sigma_2)} s'$ if, and only if $(s, \sigma_i) \in T_i$ for each $i \in \{1,2\}$, and $s' = \mathsf{jt}(s, \sigma_1, \sigma_2)$. *Runs* of a game structure are the runs of this transition system. Let $\mathcal{H}_\mathcal{T}$ denote the set of finite runs of $\mathcal{T}$. A *strategy* for Player i is a function f that maps each $h \in \mathcal{H}_\mathcal{T}$ to an action Σ_i, such that $(\mathsf{last}(h), f(h)) \in T_i$. A run is *maximal* if it is infinite (notice that a finite run can always be extended by $(\perp, \perp)$). A run ρ is *compatible* with strategies f and g of Players 1 and 2, if $\mathsf{state}_{i+1}(\rho) = \mathsf{jt}(\rho_{1\ldots i}, f(\rho_{1\ldots i}), g(\rho_{1\ldots i}))$ for all $i \geq 1$. Given strategies f and g, respectively, for Players 1 and 2, the *outcome of the pair (f,g) in $\mathcal{T}$*, denoted by $\mathsf{Outcome}_\mathcal{T}(f,g)$ is the unique maximal run that is compatible with both strategies. We denote by $\mathsf{Outcome}_\mathcal{T}(f,\cdot)$ all maximal runs compatible with f and some g, and $\mathsf{Outcome}_\mathcal{T}(\cdot,g)$ all maximal runs that are compatible with g and some f.

A game structure $\mathcal{T} = (S, s_0, \Sigma_1, \Sigma_2, T_1, T_2, \mathsf{jt})$ is *turn-based* if S can be partitioned as $S = S_1 \dot\cup S_2$ such that for any $s \in S_1$, $\sigma_1 \in \Sigma_1$, and $\sigma_2 \in \Sigma_2$, $\mathsf{jt}(s, \sigma_1, \sigma_2) = \mathsf{jt}(s, \sigma_1, \perp)$, and symmetrically for any $s \in S_2$, $\sigma_1 \in \Sigma_1$, and $\sigma_2 \in \Sigma_2$, we have $\mathsf{jt}(s, \sigma_1, \sigma_2) = \mathsf{jt}(s, \perp, \sigma_2)$. Moreover, we require that for any $s \in S_i$, $(s, \sigma) \in T_{3-i}$ if and only if $\sigma = \perp$. In other terms, in turn-based game structures, Player i determines alone the move at states S_i. We will often define turn-based game structures directly as their transition systems $(S_1 \dot\cup S_2, s_0, \Sigma, \to)$.

A *reachability objective* is a subset of states $B \subseteq S$. Player 1 has a *winning strategy* for a reachability objective B if he has a strategy f such that all runs of $\mathsf{Outcome}_\mathcal{T}(f,\cdot)$ visit B. The definition is symmetric for Player 2. A *Büchi objective* is also defined as a subset B of states. Player 1 has a *winning strategy* for a Büchi objective B, if she has a strategy f such that all runs of $\mathsf{Outcome}_\mathcal{T}(f,\cdot)$ visit B infinitely often.

2.2 Timed Automata and Games

Timed automata [AD90, AD92, AD94] are an extension of finite automata with analog clock variables, and are a convenient formalism to define timed transition systems. In this thesis, we adopt timed automata as the main formalism to model systems. We also consider their extensions *two-player timed games* [AMPS98, AMP95], which define games structures.

Given a finite set of clocks $\mathcal{C}$, we call *valuations* the elements of $\mathbb{R}^{\mathcal{C}}_{\geq 0}$. For a subset $R \subseteq \mathcal{C}$ and a valuation ν, $\nu[R \leftarrow 0]$ is the valuation defined by $\nu[R \leftarrow 0](x) = \nu(x)$ for $x \in \mathcal{C} \setminus R$ and $\nu[R \leftarrow 0](x) = 0$ for $x \in R$. Given $d \in \mathbb{R}_{\geq 0}$ and a valuation v, the valuation $\nu + d$ is defined by $(\nu + d)(x) = \nu(x) + d$ for all $x \in \mathcal{C}$. We write $\mathbf{0}$ for the valuation that assigns 0 to every clock.

An *atomic guard* is a formula of the form $k \preceq x \preceq' l$ or $k \preceq x - y \preceq' l$ where $x, y \in \mathcal{C}$, $k, l \in \mathbb{Q} \cup \{-\infty, \infty\}$ and $\preceq, \preceq' \in \{<, \leq\}$. A *guard* is a conjunction of atomic guards. A valuation ν satisfies a guard g, denoted $\nu \models g$, if all constraints are satisfied when each $x \in \mathcal{C}$ is replaced with $\nu(x)$. We denote by $[\![\phi]\!]$ the set of valuations satisfying ϕ. We write $\Phi_{\mathcal{C}}$ for the set of guards built on $\mathcal{C}$. We also define $\Phi_{\mathcal{C}}^{\leq,\geq}$ (resp. $\Phi_{\mathcal{C}}^{<,>}$) as the set of guards using only non-strict inequalities (resp. only strict equalities).

A timed automaton is a finite automaton whose edges contain guards and a clock-reset set:

Definition 2.2.1. *A* timed automaton $\mathcal{A}$ *is a tuple* $(\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E)$*, where*

- $\mathcal{L}$ *is a finite set of locations, and* $\ell_0 \in \mathcal{L}$ *is the* initial location*,*
- $\mathcal{C}$ *is a finite set of clocks,*
- Σ *is a finite set of labels,*
- $E \subseteq \mathcal{L} \times \Phi_{\mathcal{C}} \times \Sigma \times 2^{\mathcal{C}} \times \mathcal{L}$ *is a finite set of edges.*

An edge $e = (\ell, g, \sigma, R, \ell')$ *is also written as* $\ell \xrightarrow{g,\sigma,R} \ell'$*. We say that* g is the guard of the edge e*,* σ *its label, and* R *its* reset set*.*

An *open timed automaton* is a timed automaton that only uses guards in $\Phi_{\mathcal{C}}^{<,>}$. A *closed timed automaton*is one that only uses guards in $\Phi_{\mathcal{C}}^{\leq,\geq}$. We do not include for now any accepting condition in the definition of timed automata, but these will be considered in following chapters. A timed automatin is *integral* if its guards only have integer constants.

We will first present the standard semantics of timed automata, which we call the *exact semantics*. Here, clock valuations and delays are arbitrary real numbers. We will later consider different semantics, for instance, by restricting the time domain. The exact semantics of a timed automaton $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E)$ is given as a timed transition system, denoted by $[\![\mathcal{A}]\!]$, whose state space is $\mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}$, *i.e.* pairs of locations and valuations. The initial state is $(\ell_0, \mathbf{0})$; the labels are Σ, and the time domain is $\mathbb{R}_{\geq 0}$. There is a transition $(\ell, \nu) \xrightarrow{d} (\ell, \nu + d)$, for any $(\ell, \nu) \in \mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}$, and any $d \geq 0$, and $(\ell, \nu) \xrightarrow{\sigma} (\ell', \nu')$, for $\sigma \in \Sigma$, whenever $\mathcal{A}$ has an edge $(\ell, g, \sigma, R, \ell')$ with $\nu \models g$ and $\nu' = \nu[R \leftarrow 0]$. Hence, a run in a timed automaton consists in choosing delays, and edges whose guards are satisfied at current valuation. The next state is then given by resetting the designated clocks and changing location.

As an example, consider the following run of the timed automaton of Fig.1.1.

$$(\mathsf{double}, x = 0) \xrightarrow{20.5} (\mathsf{simple}, x = 20.5) \xrightarrow{\mathsf{click}} (\mathsf{simple}, x = 0) \xrightarrow{117.8}$$
$$(\mathsf{simple}, x = 117.8) \xrightarrow{\mathsf{click}} (\mathsf{simple}, x = 0) \xrightarrow{78} (\mathsf{simple}, x = 78) \xrightarrow{\mathsf{click}} (\mathsf{double}, 78)$$

Hence, the run starts at the initial state $(\mathsf{double}, \mathbf{0})$, and is an alternation of delays and actions.

Definition 2.2.2. *A* timed game *is a tuple* $(\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E_1, E_2)$ *such that* $(\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E_1 \cup E_2)$ *is a timed automaton. A* turn-based timed game *is a timed game* $(\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E_1, E_2)$ *such that* $\mathcal{L}$ *is partitioned as* $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$ *with* $\mathcal{L}_1 \cap \mathcal{L}_2 = \emptyset$*, and* $E_i \subseteq \{(\ell, g, \sigma, R, \ell') \mid \ell \in \mathcal{L}_i\}$ *for each* $i \in \{1, 2\}$.

The semantics of a timed game $\mathcal{A}$ is given as a game structure $(\mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0}, \Sigma'_1, \Sigma'_2, T_1, T_2, \mathsf{jt})$, also denoted by $[\![\mathcal{A}]\!]$. The labels are $\Sigma'_i = \mathbb{R}_{\geq 0} \times E_i \cup \{\bot\}$, that is tuples of delays and edges. The enabling conditions are defined as $T_i = (s, (d, e_i))$ with $s \in S$ and $(d, e_i) \in \Sigma'_i$ if and only if $e_i = (\ell, g, \sigma, R, \ell_1) \in E_i$ such that $s + d \models g$. The joint transition function jt obeys to the shortest delay, breaking ties in favor of Player 1. Formally, for $((\ell, \nu), (d, e_1)) \in T_1$ and $((\ell, \nu), (d', e_2)) \in T_2$, with $e_1 = (\ell, g, \sigma_1, R, \ell_1)$, and $e_2 = (\ell, g', \sigma_2, R', \ell_2)$, we let

$$\mathsf{jt}(s, (d, e_1), (d', e_2)) = \begin{cases} (\ell_1, (\nu + d)[R \leftarrow 0]) & \text{if } d \leq d', \\ (\ell_2, (\nu + d')[R' \leftarrow 0]) & \text{if } d > d'. \end{cases}$$

Furthermore, we define $\mathsf{jt}((\ell, \nu), \bot, e_2) = (\ell_2, (\nu + d')[R' \leftarrow 0])$ and $\mathsf{jt}((\ell, \nu), e_1, \bot) = (\ell_1, (\nu + d)[R \leftarrow 0])$. The initial state is $(\ell_0, \mathbf{0})$.

Notice that the semantics of a turn-based timed game is a turn-based game structure. A timed automaton can be seen as a timed game: in this case, Player 2 have no effect on the runs.

The models of timed automata and games were extended with a single cost variable, which can have different constant derivative in each location. The resulting model is called *weighted timed game (or automaton)* and was first studied in [ATP01, BFH+01].

Definition 2.2.3. *A* weighted timed game $\mathcal{A}$ *is a tuple* $(\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E_1, E_2, \mathcal{S})$ *such that* $(\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E_1, E_2)$ *is a timed game, and* $\mathcal{S} : \mathcal{L} \to \mathbb{Z}$ *is a* slope function.

The semantics of a weighted timed game is defined as that of a timed game, except that the state space also contains the value of the cost variable. We will denote the cost variable by c. Notice that this variable only "observes" the runs, and does not appear, for instance, in the guards. The slope function maps each location to an integer, which is the derivative of the cost variable when the run is at that location. Formally, the semantics of a weighted timed game $\mathcal{A}$ is given as a game structure $[\![\mathcal{A}]\!] = (\mathcal{L} \times \mathbb{R}^{\mathcal{C}}_{\geq 0} \times \mathbb{R}, \Sigma'_1, \Sigma'_2, T_1, T_2, \mathsf{jt})$, defined as for timed games. The only difference is that the states are now tuples (ℓ, ν, c), where (ℓ, ν) is a state of the underlying timed game, and c is the value of the cost variable. The joint transition function is defined as follows. For $((\ell, \nu), (d, e_1)) \in T_1$ and $((\ell, \nu), (d', e_2)) \in T_2$, with $e_1 = (\ell, g, \sigma_1, R, \ell_1)$, and $e_2 = (\ell, g', \sigma_2, R', \ell_2)$, we let

$$\mathsf{jt}((\ell, \nu, c), (d, e_1), (d', e_2)) = \begin{cases} (\ell_1, (\nu + d)[R \leftarrow 0], c + \mathcal{S}(\ell) \cdot d) & \text{if } d \leq d', \\ (\ell_2, (\nu + d')[R' \leftarrow 0], c + \mathcal{S}(\ell) \cdot d) & \text{if } d > d'. \end{cases}$$

Furthermore, we define $\mathsf{jt}((\ell, \nu, c), \bot, e_2) = (\ell_2, (\nu + d')[R' \leftarrow 0], c + \mathcal{S}(\ell) \cdot d')$ and $\mathsf{jt}((\ell, \nu), e_1, \bot) = (\ell_1, (\nu + d)[R \leftarrow 0], c + \mathcal{S}(\ell) \cdot d)$.

Notice that a weighted timed automaton can be seen as a weighted timed game in which Player 2 has no effect on the runs.

For any state (ℓ, ν) (resp. (ℓ, ν, c)) of a (weighted) timed game, let us define the projection $\mathsf{loc}((\ell, \nu)) = \ell$ (resp. $\mathsf{loc}((\ell, \nu, c)) = \ell$).

2.3 Examples

Producer-Consumer We define a simple producer-consumer system. The producer generates an event every 2 time units, and the consumer treats an event with the same period. The two components communicate through a bounded channel, modelled as an automaton. A high-level description of the system is shown in Fig. 2.1, where the buffer size is a parameter N. The system enters an error state (location err) in case of a buffer overflow.

The timed automaton $\mathcal{A}^1$ of Fig. 2.2 is the instantiation of this system for $N = 1$, where in addition, the system starts at a state $x = 1 \wedge y = 0$, which means that the consumer has an offset of 1. We also consider the timed automaton $\mathcal{A}^2$ of Fig. 2.3, where the requirements on the event periods are relaxed: *at least* one event is generated every 2 time units, and *at most* one event is consumed every 2 time units, while these two events must alternate. The timed automaton $\mathcal{A}^2$ can be seen as a more abstract specification for a system like $\mathcal{A}^1$.

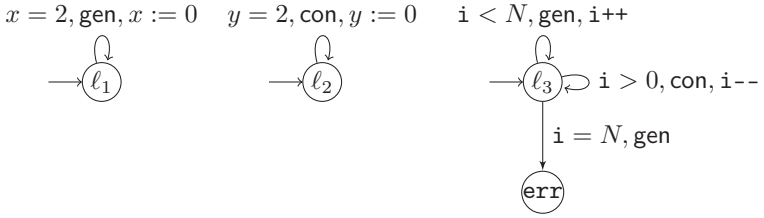


Figure 2.1: A timed automaton modelling a producer-consumer system, given as a *network of timed automata* with a bounded integer variable i which is initially 0. Here, each component is a separate timed automaton. Time progresses globally, and any action shared between two components is only possible if both components move simultaneously. Networks of timed automata, and finite variables allow a more compact representation. We did not formally define these, but they can be easily encoded in the underlying finite automaton. In our discussion, we concentrate on an instantiation for $N = 1$, given explicitly as a timed automaton in Fig. 2.2.

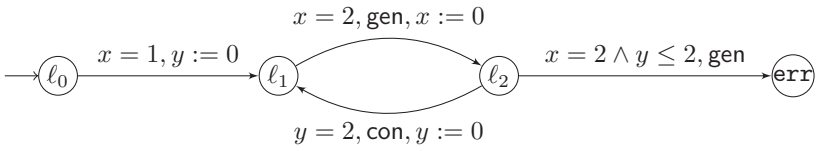


Figure 2.2: The timed automaton $\mathcal{A}^1$ modelling the producer-consumer system of Fig. 2.1 instantiated for $N = 1$. The transition from ℓ_0 to ℓ_1 models an offset of 1 time unit given for the consumer.

Scheduling An application of timed automata is the synthesis of schedulers in various contexts [AAM06]. Let us give here an encoding of an instance of the *job-shop scheduling* problem.

Consider the scheduling problem instance described in Fig. 2.4, which is a variant of an example of [RWT+06]. Assume that we look for a *work-conserving* scheduler, that will immediately start executing a task if a machine is free for execution and a task is available. What execution time can

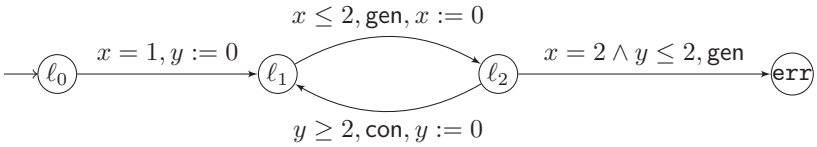


Figure 2.3: A timed automaton $\mathcal{A}^2$ specifying the behavior of the producer-consumer system of Fig. 2.2. Here, the specification is more abstract: an allowed behavior consists in generating at least an event every 2 time units, and consuming at most one every 2 time units, while alternating between generation and consumption. This automaton coincides with the example given by Puri [Pur00].

a work-conserving scheduling policy guarantee on this instance? One can model this problem as a timed automaton, and prove, by reachability analysis, that these tasks can be scheduled using a work-conserving policy within six time units. In fact, the scheduling given in Fig. 2.4 is a feasible instance.

The dependences, the work-conserving policy and other constraints can be encoded as a timed automaton, where clocks are used to count execution time. To check whether tasks can be scheduled in c time units, one can use an additional clock t that measures the time elapsed from the beginning, and require $t \leq c$ at every edge. Let us describe the timed automaton encoding this problem. For convenience, we define the system as a network of timed automata with synchronization on shared actions, and we use boolean variables. As previously, the corresponding timed automaton (conforming to Definition 2.2.1) can be deduced easily.

The system contains two processes: Process 1 executing A and B, and Process 2 executing C, D, and E. Both processes need the same resources (machines M_1 and M_2), which can only be used exclusively. This system can be modelled by a network of timed automata, as follows. We define a timed automaton $\mathcal{P}_1$ corresponding to Process 1, and $\mathcal{P}_2$ corresponding to Process 2. A third timed automaton $\mathcal{M}$ enforces the mutual exclusion for both machines: no more than one task is executed on a machine at any time. Moreover, we will also use $\mathcal{M}$ to impose a work-conserving scheduling: this automaton will block any action if some task is waiting for execution on a machine that is free (more details follow below). Each automaton $\mathcal{P}_i$ makes sure that its tasks are executed respecting the dependences and timing constraints. All three components are given in Figures 2.5–2.7. For instance, in automaton $\mathcal{P}_1$, the task A can be executed on M_1 (first edge), only if lock_1 can be taken. The lock is released upon the termination of the task (second edge). Automaton $\mathcal{M}$ has two boolean variables b_1, b_2, where b_i is set to true if machine M_i is busy. The two self-loops are

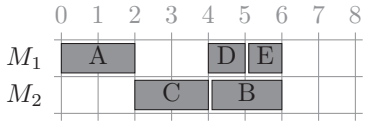


Figure 2.4: Consider tasks A, B, C of duration 2 and D, E of duration 1. Dependences between tasks are as follows: $A \to B$ and $C \to D, E$, meaning *e.g.* that A must be completed before B can start. Task A must be executed on machine M_1 and tasks B, C on machine M_2. Moreover, task C cannot be scheduled before 2 time units (which could be modelled using an extra task). The figure shows an optimal work-conserving schedule for these tasks under these constraints.

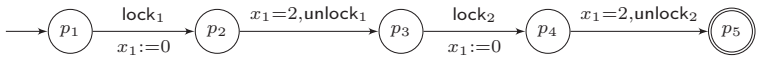


Figure 2.5: Timed automaton $\mathcal{P}_1$ modeling Process 1.

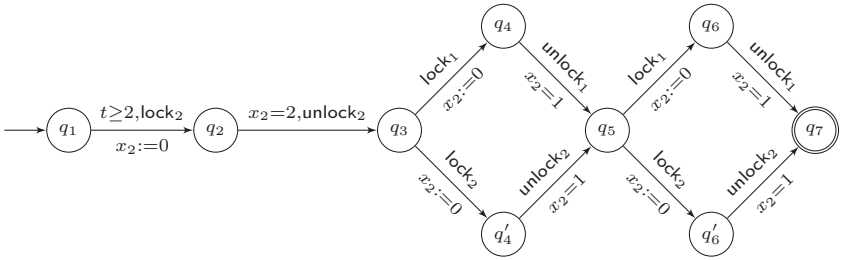


Figure 2.6: Timed automaton $\mathcal{P}_2$ modeling Process 2.

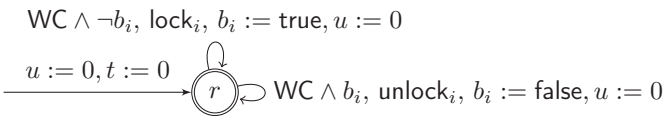


Figure 2.7: Timed automaton $\mathcal{M}$ ensuring mutual exclusion and greedy scheduling. The guard WC ensures the work-conserving property: $\mathsf{WC} = (\forall i \in \{1,2\}, \neg b_i \wedge \mathsf{Waiting}_i \Rightarrow u = 0) \wedge t \leq c$. Here t is a clock that is never reset. The constraint $t \leq c$ ensures that we only consider schedules that terminate in c time units.

defined for $i = 1, 2$. For instance, the automaton $\mathcal{M}$ synchronizes with action lock_i only if b_i is not set to true. It has the guard $\forall i \in \{1,2\}, \neg b_i \wedge \mathsf{Waiting}_i \Rightarrow u = 0$ and $t \leq c$ (the universal quantifier and implication are only used here to obtain a compact representation; an equivalent guard can be obtained without difficulty). The first one ensures work-conserving scheduling, while the second one blocks any action after c time units. Here, c needs to be instantiated for the timed automaton to be defined; one could choose $c = 6$ for our purpose. Here, $\mathsf{Waiting}_i$ is a formula stating that some task is ready for execution on machine i. We omit the definition of $\mathsf{Waiting}_i$ here but it can be easily defined using the locations of components $\mathcal{P}_1$ and $\mathcal{P}_2$. Automaton $\mathcal{M}$ blocks any action in the whole system if time has elapsed while some task was ready for execution on a machine that is free. Notice also that we marked some locations as *final* in each component. This means that we are only interested in runs that end in final locations in all components as these correspond to successful scheduling of all tasks.

Scheduling with Interruption Timed games can be used to model scheduling instances where scheduled tasks are cancelled by an external event. A simple example is given in Fig. 2.8. In this example, the location p_1 represents the scheduling of a task that takes 3 time units to execute, and the objective of Player 1 is to reach the location p_2 in at most 4 time units. However, before the guard $x = 3$ is enabled, Player 2 can "interrupt" the execution and move the system to p_3. Here, the task needs to be scheduled once again, but the objective is only reached if there remains enough

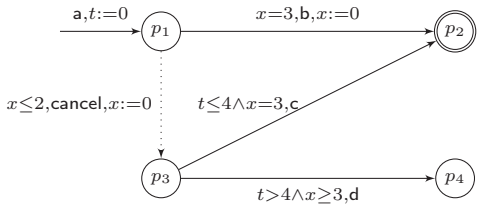


Figure 2.8: A timed game. Player 1's edges are represented as straight edges, while Player 2's edges are dashed.

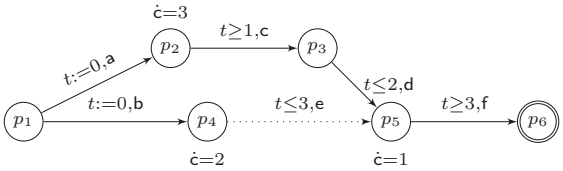


Figure 2.9: A weighted timed game.

time (if $t \leq 4 \wedge x = 3$ is reachable). Otherwise, the system is stuck at p_3 or p_4.

Weighted Timed Games A cost variable can be added to timed automata and games to express varying amounts of resource. A scheduling problem similar to ones above is given in Fig. 2.9, where, moreover, several executions with different costs are possible. In this weighted timed game, the goal of Player 1 is to reach the location p_6. Roughly, two kinds of strategies allow reaching p_6: one can first go either to p_2 or to p_4. In the former case, the cost grows with derivative 3 at location p_2 where at least one time unit must be spend. In the latter case, the cost grows with a smaller derivative (namely, 2) in location p_4; however the time spent there is determined by Player 2. In either case, the play must stay at p_5 where the cost grows with derivative 1, until $t \geq 3$. Here, the least cost that Player 1 can guarantee while reaching p_6 is 4. In fact, she can go to p_2 and spend exactly one time unit, while the cost grows by 3, then, spend one time unit in p_3, and one time unit in p_5, where the grows by 1. The path through p_4 does not yield a better cost, since Player 2 can wait 3 time units in location p_4, which yields a cost of at least 6.

2.4 Perturbation Models

In order to study robustness in timed automata, one first needs to define a perturbation model against which a given model is to be checked. We present and compare several perturbed semantics we study in this thesis, and illustrate these on the examples of the previous section. A common feature of all semantics is that we consider that the bound on the imprecisions (denoted by δ below), or the sampling period (denoted $\frac{1}{k}$) are seen as unknown parameters to be synthesized by robustness analysis. In fact, if an exact value for this parameter is known, one can often encode the resulting semantics by a timed automaton, although such an encoding often increases the size of the model. Here, we are rather interested in understanding the timed automata that are robust for all values of

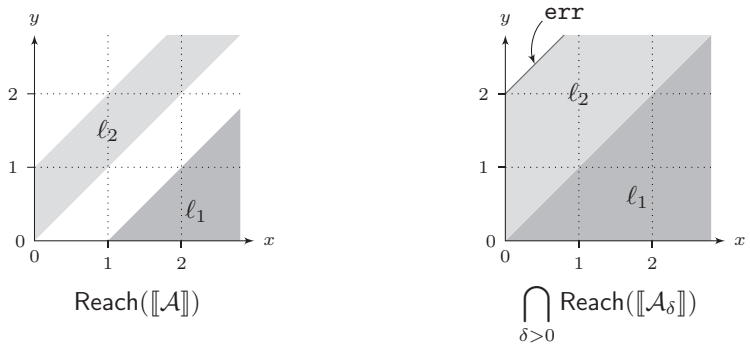


Figure 2.10: The set of reachable states of $\mathcal{A}^2$ (on the left) and $\mathcal{A}^2_\delta$ for an arbitrary $\delta > 0$ (on the right). The location `err` is not reachable in $\mathcal{A}^2$ but becomes reachable in $\mathcal{A}^2_\delta$.

the parameter under *some* bound, and in synthesizing this bound to ensure correctness.

Note that some of the following semantics are only defined for timed automata, since we do not study all semantics in (weighted) timed games.

2.4.1 Enlargement

Given a parameter $\delta > 0$, *enlargement* consists in replacing each guard of the form $x \in [a, b]$ by $x \in [a - \delta, b + \delta]$. Formally, given a clock set $\mathcal{C}$, we define the enlargement of an atomic guard g by δ, denoted $\langle g \rangle_\delta$, as follows.

$$\langle k \preceq x - y \preceq' l \rangle_\delta = k - \delta \preceq x - y \preceq' l + \delta,$$
$$\text{and} \qquad \langle k \preceq x \preceq' l \rangle_\delta = k - \delta \preceq x \preceq' l + \delta.$$

for any $x, y \in \mathcal{C}$, $k, l \in \mathbb{Q}$ and $\preceq, \preceq' \in \{<, \leq\}$. The enlargement of a guard is obtained by enlarging its atomic guards. Given a timed automaton $\mathcal{A}$, let us denote by $\mathcal{A}_\delta$ the timed automaton obtained from $\mathcal{A}$ by enlarging by δ all its guards.

It is easy to show that any behavior of $\mathcal{A}$ is still present in $\mathcal{A}_\delta$ for any $\delta > 0$, in the sense that $[\![\mathcal{A}]\!] \sqsubseteq [\![\mathcal{A}_\delta]\!]$, where the identity relation can be used to show that simulation holds. Hence, the enlargement consists in adding new behaviors by means of relaxing the guards by some parameterized amount. Robustness in this context means that a timed automaton should satisfy a specification even if the guards are enlarged by a small $\delta > 0$. In other terms, a slight error in the timings should not lead the system to an error. It was shown in [DDMR08] that the timed automaton $\mathcal{A}^2$ of Fig. 2.2 is not robust to guard enlargements, for any value of $\delta > 0$, in the following sense. While the location `err` is not reachable in the exact semantics, it becomes reachable in $\mathcal{A}^2_\delta$ for all $\delta > 0$. Hence, the slightest enlargement changes the qualitative behavior of the timed automaton. The sets of reachable states, for $\delta = 0$ and $\delta > 0$, are illustrated in Fig. 2.10. Note that the timed automaton $\mathcal{A}^1$ also suffers from the same robustness problem: the location `err` becomes reachable under any positive enlargement.

Enlargement has also implications in *implementability* of timed automata, that is, whether one can make sure that the implemented system preserves the exact semantics of the model. In [DDR05a], the authors define a *program semantics*, corresponding to the execution of timed automata by a simple model of microprocessor with an imprecise clock, a buffer and a positive reaction time. It was

shown that when parameters are chosen appropriately, the enlargement yields an overapproximation of the program semantics. This motivates the study of enlargement in timed automata since one can prove the correctness of a system executed on a microprocessor, in this framework.

Chapter 4 treats the robustness analysis against guard enlargements, while Chapter 11 presents a variant of the program semantics. Enlargement is considered throughout this thesis as a general technique.

2.4.2 Shrinking

One might also be interested in the dual notion of enlargement, that is checking whether the semantics of a given timed automaton is vulnerable to *guard shrinkings*, which consists in transforming a guard $x \in [a, b]$ into $x \in [a + \delta, b - \delta']$. More precisely, we will be interested in checking whether the guards of a given timed automaton can be shrunk so that the resulting timed automaton time-abstract simulates the original timed automaton, and is non-blocking. So, we are asking whether the behavior of the timed automaton depends on the system's ability to take transitions at the borders of the guards (*i.e.* at the last moment, or immediately after enablement). We believe that the behavior of a good timed automaton design should not change when one forbids such "limit" behavior.

Formally, we define the shrinking of an atomic guard g as its enlargement by a negative amount. We will consider possibly different shrinking parameters for each atomic guard in a timed automaton. So for any timed automaton $\mathcal{A}$, let I denote the set of its atomic guards. Given a vector δ of nonnegative rational numbers indexed by I, let us denote by $\mathcal{A}_{-\delta}$ the timed automaton obtained from $\mathcal{A}$ by shrinking each atomic guard $i \in I$ by δ_i. We will rather choose these parameters as $\mathbf{k}\delta$, where $\mathbf{k}$ is a set of natural numbers, and δ is a rational. Note that any finite vector of nonnegative rational numbers can be written in this form.

We will be interested in the synthesis of these parameters $\mathbf{k}\delta$ so as to ensure time-abstract simulation and non-blockingness of the resulting timed automaton. Let us show that the timed automaton $\mathcal{A}^2$ of Fig. 2.3 is not *shrinkable*, in the sense that any shrinking yields a timed automaton that cannot time-abstract simulate $\mathcal{A}^2$. The cycle $(\mathsf{gen}\cdot\mathsf{con})^\omega$ is a possible behavior of $[\![\mathcal{A}^2]\!]$. However, if one of the guards is shrunk, say, if we replace $x \leq 2$ by $x \leq 2 - \delta$, then, such a cycle can only be taken at most $\frac{1}{\delta}$ times. In fact, one can show that the value of x is increased by at least δ at each iteration, so any run will eventually reach a deadlock. This means that the behavior $(\mathsf{gen} \cdot \mathsf{con})^\omega$ is only possible if the system takes each action "at the last moment", right before the guard is disabled. This behavior is therefore not "robustly realizable".

Shrinking is also a natural method for refining or implementing a timed automaton under imprecisions. In fact, if one "implements" a guard $x \in [a, b]$ as $x \in [a + \delta, b - \delta]$, then, under imprecisions bounded by some Δ, the system guarantees that $x \in [a + \delta - \Delta, b - \delta + \Delta]$. Provided that $0 \leq \Delta < \delta$, the resulting behavior conforms to the initial model since

$$[a + \delta - \Delta, b - \delta + \Delta] \subseteq [a, b].$$

Thus, no new behavior is present in the implemented system under imprecisions. Formally, one can show that $[\![\mathcal{A}_{-\mathbf{k}\delta+\Delta}]\!] \sqsubseteq [\![\mathcal{A}]\!]$ for appropriately chosen $\mathbf{k}$, δ and Δ.

Shrinkability analysis is studied in Chapter 5. We discuss the technique of implementation by shrinking in Chapter 11.

2.4.3 Sampling

Timed automata are a generic formalism that allows the design of various systems. It is sometimes convenient to use timed automata, and its continuous time semantics, to model a system that is intended to be implemented under sampled time. The *sampled semantics* of a timed automaton $\mathcal{A}$ consists in defining the TTS as in the exact semantics, except that the time domain is $\frac{1}{k}\mathbb{N}$ for some $k \geq 1$. This semantics is denoted by $[\![\mathcal{A}]\!]^{\frac{1}{k}}$.

It should be clear that any behavior of the sampled semantics is a behavior of the (continuous-time) exact semantics. The converse does not always hold, since some infinite runs may require delays with arbitrary precisions [CHR02]. For instace, in timed automaton $\mathcal{A}^2$, if we replace the guards by strict counterparts, that is $x \leq 2$ by $x < 2$, and $y \geq 2$ by $y > 2$, then the cycle $(\mathsf{gen} \cdot \mathsf{con})^\omega$ cannot be repeated infinitely often under any sampling. Robustness against sampling consists in checking whether the sampled semantics preserves some aspects of the exact semantics.

When the sampling period is seen as a parameter, the parameterized safety problem is undecidable [CHR02], while the parameterized reachability problem is decidable [AKY07]. Untimed language equivalence was shown decidable in [AKY10].

In Chapter 10, we give a construction to systematically implement a given timed automaton under the sampled semantics, while the behavior is approximately preserved.

2.4.4 Game Semantics: Excess-Perturbation Game

The semantics we defined above are interesting for worst-case analysis. For instance, when checking safety properties under guard enlargement, one seeks to make sure that a bad behavior is not possible under any scenario, given this perturbation. However, such an analysis does not take into account the fact that a real-time controller can observe the perturbations happened in the past, and try to adapt its behavior to reach a desired state or avoid a bad state. We define here a game semantics to model robustness while capturing the possible reaction of the system.

Given a timed game $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \Sigma_1, \Sigma_2, E_1, E_2)$ and a parameter $\delta > 0$, we define the *excess-perturbation game* of $\mathcal{A}$ w.r.t. δ as a game structure $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ between players *Controller* and *Perturbator*. Intuitively the semantics is the following: At any location, Controller and Perturbator both suggest an action, which is either a delay and an edge, or the empty action $\perp$. If Perturbator's delay is shorter, the suggested delay and action are taken. Otherwise, the play moves to an intermediate state where Perturbator chooses a perturbation among $[-\delta, \delta]$. Then, the delay and the edge suggested by Controller are taken after this perturbation. Formally, the state space of $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ is $S = \mathcal{L} \times \mathbb{R}_{\geq 0}^{\mathcal{C}} \cup \mathcal{L} \times \mathbb{R}_{\geq 0}^{\mathcal{C}} \times \mathbb{R}_{\geq 0} \times E$, and the initial state is $(\ell_0, \mathbf{0})$. The labels are $\Sigma_1' = \mathbb{R}_{\geq 0} \times E_1 \cup \{\perp\}$ and $\Sigma_2' = \mathbb{R}_{\geq 0} \times E_2 \cup [-\delta, \delta] \cup \{\perp\}$. The enabling condition T_1 for Controller is as follows. From any state $(\ell, \nu) \in S$, we have $((\ell, \nu), d, e_1) \in T_1$ for any $d \geq \delta$ and $e_1 \in E_1$ such that if we write $e_1 = (\ell, g, \sigma_1, R, \ell_1)$, then $\nu + d \models g$. For states $(\ell, \nu, d, e) \in S$, we have $((\ell, \nu, d, e), \perp) \in T_1$. For Perturbator, the enabling condition T_2 are as follows. From any state $(\ell, \nu) \in S$, we have $((\ell, \nu), d, e_2) \in T_2$ for any $d \geq \delta$ and $e_2 \in E_2$ such that if we write $e_2 = (\ell, g', \sigma_2, R', \ell_2)$, then $\nu + d \models g$[1] For states $(\ell, \nu, d, e) \in S$, we have $((\ell, \nu, d, e), \epsilon) \in T_2$ for any $\epsilon \in [-\delta, \delta]$. The joint transition function δ respects the shorter delay, as in timed games. We let

$$\mathsf{jt}((\ell, \nu), (d, e_1), (d', e_2)) = \begin{cases} (\ell, \nu, d, e_1) & \text{if } d \leq d', \\ (\ell_2, (\nu + d')[R' \leftarrow 0]) & \text{if } d > d'. \end{cases}$$

[1] In Chapter 7, we will consider a variant which does not impose a lower bound on Perturbator's delays. This choice corresponds to different assumptions on the environment.

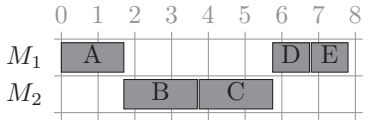


Figure 2.11: The scheduling instance of Fig. 2.4 where the duration of task A is reduced to $2-\delta$. Because of the work-conserving policy, the optimal scheduling is now as shown in this figure, and takes $8-\delta$ time units. This is a timing anomaly that is captured by the excess-perturbation semantics.

We define $\mathsf{jt}((\ell,\nu),(d,e_1),\bot) = (\ell,\nu,d,e_1)$ and $\mathsf{jt}((\ell,\nu),\bot,(d',e_2)) = (\ell_2,(\nu+d')[R'\leftarrow 0])$, using the same notations as above. Last, we let $\mathsf{jt}((\ell,\nu,d,e_1),\bot,\epsilon) = (\ell_1,(\nu+d+\epsilon)[R\leftarrow 0])$.

The semantics for a weighted timed game $\mathcal{A} = (\mathcal{L},\ell_0,\mathcal{C},\Sigma_1,\Sigma_2,E_1,E_2,\mathcal{S})$ is very similar. The state space now contains, in addition, a real value for the cost variable. We only need to define the change in the cost variable. At any state (ℓ,ν,c), given two actions $(d,e_1)\in\Sigma_1'$ and $(d',e_2)\in\Sigma_2'$, with $e_1 = (\ell,g,\sigma_1,R,\ell_1)$ and $e_2 = (\ell,g',\sigma_2,R',\ell_2)$, we let

$$\mathsf{jt}((\ell,\nu,c),(d,e_1),(d',e_2)) = \begin{cases} (\ell_1,\nu,c,d,e_1) & \text{if } d\le d',\\ (\ell_2,(\nu+d')[R'\leftarrow 0],c+d'\cdot\mathcal{S}(\ell)) & \text{if } d>d'.\end{cases}$$

Further, for any state (ℓ,ν,d,e_1), we have

$$\mathsf{jt}((\ell,\nu,d,e_1),\bot,\epsilon) = (\nu+d)[R\leftarrow 0],c+(d+\epsilon)\cdot\mathcal{S}(\ell)).$$

Hence the cost has derivative $\mathcal{S}(\ell)$ at location ℓ during the perturbed delay.

In this definition, we restrict Controller to play delays that are at least δ. This models the fact that consecutive actions must be separated by a positive amount of time. We assume for simplicity that this lower bound and the magnitude of the perturbation are both δ. They could be chosen as different parameters; our results presented in Part III would hold. The name excess-perturbation comes from the fact that although Controller is required to suggest delays and edges that satisfy the guard, Perturbator's moves are not required to respect the guards. We also consider a conservative variant of this semantics where guards must be satisfied under any perturbation. See below for a comparison of the two semantics.

Let us consider the scheduling example of Figures 2.5–2.7. If we consider the excess-perturbation semantics, then, while Controller can only suggest a delay of 2 for the termination of task A (between locations p_2 and p_3), the actual delay could be shorter or longer by δ. Assume Perturbator chooses to delay $2-\delta$. Then, Figure 2.11 shows the optimal scheduling (that is, with least completion time) after the termination of task A, which takes $8-\delta$ time units. Thus, any small decrease in the execution time of A increses the completion time by almost 2 time units. This is a *timing anomaly*, a well-known phenomenon in scheduling [RWT$^+$06].

The excess-perturbation semantics can capture such timing anomalies in scheduling applications, and ensure robust control in other applications. In fact, here, Controller doesn't have a strategy to reach the final location under the constraint $t\le 6$, for any $\delta>0$. But one can show that for any $c>8$, it has a strategy under the constraint $t\le c$, for small enough $\delta>0$.

We consider the parameterized robust reachability problem in this semantics in Chapter 7.

2.4.5 Game Semantics: Conservative-Perturbation Game

Another possibility when defining a game semantics to capture robustness is to require that Controller should suggest moves that satisfy the guard of the chosen edge under *any* move of Perturbator. This means restricting Controller to play far from the borders of the guard. In particular, equality constraints are never enabled in such a semantics. This semantics is useful if one already accounts for imprecisions in the model, by providing time intervals for each action. Then, the designer is interested in staying strictly inside the guards, rather than analyzing additional runs. This corresponds to a different design approach (see below for a comparison with the previous semantics).

Formally, we define the *conservative perturbation game* for a given weighted timed game $\mathcal{A}$, denoted by $\mathcal{G}_\delta^{\mathsf{cons}}(\mathcal{A})$, similarly to $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ with only the following difference. The enabling condition T_1 for Controller is modified as $((\ell,\nu),d,e_1) \in T_1$ for any $d \geq \delta$ and $e_1 \in E_1$ such that if we write $e_1 = (\ell, g, \sigma_1, R, \ell_1)$, then $\nu + d + \epsilon \models g$ for all $\epsilon \in [-\delta, \delta]$.

When studying the behavior under this semantics, the emphasis is no more on the newly appeared behaviors, as this is the case in the excess-perturbation game semantics. In fact, it follows from the definition that any outcome of this game is a run of the exact semantics of the timed automaton. One is rather interested in the loss of behaviors. For instance, equality constraints or combination of other guards that imply such a constraint are never enabled. But what is more interesting is that some infinite runs may be disabled under this semantics. For instance, the cycle $(\mathsf{gen} \cdot \mathsf{con})^\omega$ cannot be repeated an infinite number of times under some strategy of Perturbator. In chapter 8, we will be interested in deciding whether one can ensure infinite runs satisfying some Büchi conditions, for some value of δ.

This semantics with a known value for δ was studied in [CHP11] for timed games with parity conditions. In this case, using an appropriate encoding, the problem can be reduced to solving usual timed games. The results of this thesis also shows that this semantics is related to recent work on convergence phenomena on timed automata runs [CHR02, BA11, Sta12].

Excess or Conservation? We defined here two very similar game semantics. We believe both are meaningful and can appear naturally at different levels of abstraction. For instance, the excess-perturbation game semantics is a natural choice when modelling a real-time system with fixed task execution times, if we also know that these execution times will be subject to perturbation whose magnitude is unknown in advance, which may depend on the implementation platform to be chosen later. Incorporating these perturbations in the model, for instance, by replacing equality constraints by non-punctual intervals, requires a choice of a suitable interval length which may not be known. Moreover this will increase the state space in general. Hence, as in the scheduling example of Section 2.3, the excess-perturbation game semantics allows one to keep the design abstract and still apply robustness analysis.

On the other hand, in some applications, specifying distinct lower and upper bounds in timings may be natural. For instance, if one is interested in modelling an embedded system that should send a signal to another component which accepts input signals in a time interval $[l, u]$, then it is natural to look for a controller that strictly respects this interval. Such a model and its semantics would be closer to the actual program. In this case, the conservative-perturbation game semantics is the natural choice.

Chapter 3

Data Structures

In this chapter, we present several data structures and some technical results used throughout this thesis. We start by defining *regions* and *region automata*, which are well known notions in the timed automata literature [AD94], upon which several results have been built. Among these, we define here *orbit graphs*, which are graphs that summarize the reachability relations between the vertices of regions [Pur00]. Then, we introduce an original data structure, called *shrunk DBM*, which is an extension of difference-bound matrices used to represent "shrunk zones". We give a parameterized and a non-parameterized version of this data structure.

3.1 Regions

The decidability of several verification problems in timed automata relies on the fact that a finite-index quotient of the state space can be defined while preserving the important behaviors. Formally, given a timed automaton $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E)$, let M denote the largest constant that appears in its guards in absolute value. We define an equivalence relation $\simeq_M$ on valuations as follows. For any $\nu, \nu' \in \mathbb{R}^{\mathcal{C}}_{\geq 0}$, we let $\nu \simeq_M \nu'$ if, and only if the following conditions are satisfied, for all pairs of clocks $x, y \in \mathcal{C}$.

1. either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or $\nu(x), \nu'(x) > M$,
2. if $\mathsf{frac}(\nu(x)), \mathsf{frac}(\nu'(x)) \leq M$, then $\mathsf{frac}(\nu(x)) = 0 \;\Leftrightarrow\; \mathsf{frac}(\nu(x)') = 0$,
3. if $\mathsf{frac}(\nu(x)), \mathsf{frac}(\nu(y)) \leq M$, then
$$\mathsf{frac}(\nu(x)) \leq \mathsf{frac}(\nu(y)) \;\Leftrightarrow\; \mathsf{frac}(\nu'(x)) \leq \mathsf{frac}(\nu'(y)),$$
4. for any interval I among $(-\infty, -M), [-M, M], (-M, -M+1), \ldots, (-1, 0), [0, 0], (0, 1), \ldots, [M, M], (M, \infty)$, we have,
$$\nu(x) - \nu(y) \in I \Leftrightarrow \nu'(x) - \nu'(y) \in I.$$

where $\mathsf{frac}(\cdot)$ denotes the fractional part. The equivalence class of a valuation ν for the relation $\simeq$ is denoted by $\mathsf{reg}(\nu) = \{\nu' \mid \nu \simeq_M \nu'\}$, and called a *$M$-region* of $\mathcal{A}$. When M is clear from context, these are simply called regions. It is known that $\simeq_M$ has finite index [AD94], which can be exponential in the size of $\mathcal{A}$.

Our definition differs from the first definition of regions of [AD94]; the fourth condition appears later in [BDFP00]. In most chapters, we will assume that clocks do not go above the maximum constant. Under this assumption, both definitions are equivalent. We only need the fourth condition in Chapter 10, where we also give a more refined definition of regions using arbitrary *granularity* for the constants of the guards.

Assuming that clocks do not go above some constant means that we discard all runs containing too long delays, or along which some clocks are not reset. This assumption appears in several works *e.g.* [Pur00, DDMR08]. Let us call *bounded regions*, those regions where all clocks are below the constant M. Any bounded region r is defined by fixing the integer parts of the clocks, and giving a partition $X_0, X_1, \ldots, X_m$ of the clocks, ordered according to their fractional values: for any $\nu \in r$, $0 = \mathsf{frac}(\nu(x_0)) < \mathsf{frac}(\nu(x_1)) < \ldots < \mathsf{frac}(\nu(x_m))$ for any $x_0 \in X_0, \ldots, x_m \in X_m$, and $\mathsf{frac}(\nu(x)) = \mathsf{frac}(\nu(y))$ for any $x, y \in X_i$. Here, $X_i \neq \emptyset$ for all $1 \leq i \leq m$ but X_0 can be empty. We will refer to this partition by *the partition according to the fractional ordering*. Figure 3.1 contains examples of regions with two clocks.

The *region automaton* of $\mathcal{A}$ is a finite automaton with alphabet $\Sigma \dot\cup \{\mathsf{delay}\}$, denoted by $\mathcal{R}(\mathcal{A})$, defined as follows [AD94]. The states are pairs (ℓ, r) where $\ell \in \mathcal{L}$ and r is a region. There is a transition $(\ell, r) \xrightarrow{\mathsf{delay}} (\ell, r')$ if, and only if, $\nu' = \nu + d$ for some $\nu \in r$, $\nu' \in r'$ and $d \geq 0$. There is a transition $(\ell, r) \xrightarrow{\sigma} (\ell', r')$ if, and only if $\mathcal{A}$ has an edge $(\ell, g, \sigma, R, \ell')$, and for some $\nu \in r$, writing $\nu' = \nu[R \leftarrow 0]$, we have $r' = \mathsf{reg}(\nu')$. A region automaton is thus a finite transition system over alphabet $\mathsf{E} \cup \{\mathsf{delay}\}$. We will call *paths* the runs of this transition system. Given a run ρ, its *projection on regions* is the path π in the region automaton s.t. $\mathsf{state}_i(\rho) \in \mathsf{state}_i(\pi)$ for all $1 \leq i \leq n$, and either $\mathsf{trans}_i(\rho) = \mathsf{trans}_i(\pi)$ or $\mathsf{trans}_i(\pi) = \mathsf{delay}$ and $\mathsf{trans}_i(\rho) \in \mathbb{R}_{\geq 0}$. In this case, we write $\mathsf{first}(\rho) \xrightarrow{\pi} \mathsf{last}(\rho)$ (and say that ρ is *along* π). A *lasso* is a path $\pi_0\pi_1$ where π_1 is a cycle, *i.e.* $\mathsf{first}(\pi_1) = \mathsf{last}(\pi_1)$. A *cycle* of $\mathcal{R}(\mathcal{A})$ is a *progress cycle* if it resets all clocks at least once [Pur00].

We define the *timed-action region automaton* $\mathcal{R}_{\mathrm{ta}}(\mathcal{A})$, similarly to the region automaton, on the set of pairs (ℓ, r) of locations and regions, with the difference that there is a transition $(\ell, r) \xrightarrow{\sigma} (\ell', r')$ if, and only if for some $\nu \in r$, and $\nu' \in r'$, $d \geq 0$ and an edge $(\ell, g, \sigma, R, \ell')$, $\nu' = (\nu + d)[R \leftarrow 0]$. Hence, delays are now hidden inside actions. We define the paths of $\mathcal{R}_{\mathrm{ta}}(\mathcal{A})$ similarly to $\mathcal{R}(\mathcal{A})$. Given a run ρ of $\mathcal{A}$, its *timed-action projection on regions* is the path π in the timed-action region automaton s.t. $\mathsf{state}_i(\rho) \in \mathsf{state}_i(\pi)$ for all odd $1 \leq i \leq n$, and $\mathsf{trans}_i(\rho) = \mathsf{trans}_i(\pi)$ for even $1 \leq i \leq n$.

It is known that $\mathcal{R}(\mathcal{A})$ is time-abstract bisimilar to $\mathcal{A}$, whereas $\mathcal{R}_{\mathrm{ta}}(\mathcal{A})$ is weakly time-abstract bisimilar (weak bisimulation simply does not observe delays). Hence, any time-abstract property on $\mathcal{A}$ can be checked on $\mathcal{R}(\mathcal{A})$ or $\mathcal{R}_{\mathrm{ta}}(\mathcal{A})$. These automata have exponential size. Some minimization algorithms have been studied in order to construct a minimized version of $\mathcal{R}(\mathcal{A})$ [TY01].

3.2 Orbit graphs

A *vertex* of a region r is any point of $\bar{r} \cap \mathbb{N}^{\mathcal{C}}$, where $\bar{r}$ denotes the topological closure of r. For any region r, and any clock x, let us denote by $r_{x,0}$ the upper bound (by $-r_{0,x}$ the lower bound) on clock x inside region r. Consider the partition of the clocks $X_0, X_1, \ldots, X_m$ according to their fractional parts in r. Any vertex v of r is defined by the choice of an index $1 \leq i \leq m$ such that for all $x \in X_1, \ldots, X_i$, we have $v(x) = -r_{0,x}$ and for all $x \in X_{i+1}, \ldots, X_m$, we have $v(x) = r_{x,0}$. Hence, any region has at most $|\mathcal{C}| + 1$ vertices (See also [DDMR08]). Let $\mathcal{V}(r)$ denote the set of vertices of r. We also extend this definiton to $\mathcal{V}((\ell, r)) = \mathcal{V}(r)$. It has been shown that all valuations in r are convex combinations of $\mathcal{V}(r)$ [Pur00].

To any path π of the region automaton, we associate a $|\pi| + 1$-partite graph $\gamma(\pi)$ called the *orbit graph of* π [Pur00]. Intuitively, the orbit graph of a path gives the reachability relation between the vertices of the regions visited along the path. Formally, for a transition $\tau = q_1 e_1 q_2$, $\gamma(\tau) = (V_1 \cup V_2, f_G, E)$ is a bipartite graph where $V_1 = \{(1, v)\}_{v \in \mathcal{V}(q_1)}$, and $V_2 = \{(2, v)\}_{v \in \mathcal{V}(q_2)}$. For any $\big((1, u), (2, v)\big) \in V_1 \times V_2$, we have an edge $(u, v) \in E$, if, and only if $u \xrightarrow{\overline{e_1}} v$, where $\overline{e_1} = \mathsf{delay}$ if

$e_1 = \mathsf{delay}$, and otherwise $\overline{e_1}$ is obtained by replacing the guard by its closed counterpart. Note that each vertex has at least one successor through $\overline{e_1}$ since the guard is closed; this follows from the properties of regions [AD94]. The labelling function f_G maps each vertex of V_i to q_i. In order to extend $\gamma(\cdot)$ to paths, we use a composition operator $\oplus$ between orbit graphs, defined as follows. If $G = (V_1 \cup \ldots \cup V_n, f_G, E)$ and $G' = (V'_1 \cup \ldots \cup V'_m, f_{G'}, E')$ denote two orbit graphs, then $G \oplus G'$ is defined if, and only if $f_G(V_n) = f_{G'}(V'_1)$. In this case, the graph $G'' = G \oplus G' = (V''_1 \cup \ldots V''_{n+m-1}, f_{G''}, E'')$ is defined by taking the disjoint union of G and G', merging each node (n, v) of V_n with the node $(1, v)$ of V'_1, and renaming any node $(i, v) \in V'_i$ by $(i+n-1, v)$, so that we get a $(n+n'-1)$-partite graph. Formally, we let $V_i = V''_i$ for all $1 \leq i \leq n$, and the subgraph of G'' induced on these nodes is equal to G. For any $n+1 \leq i \leq n+m-1$, we have $V''_i = \{(i, v)\}_{(i-n+1,v) \in V'_{i-n+1}}$, and there is an edge $\big((i, v), (i+1, w)\big) \in E''$ if, and only if $\big((i-n+1, v), (i-n, w)\big) \in E'$. Now, we extend $\gamma(\cdot)$ to paths by induction, as follows. Consider any $\pi = q_1 e_1 \ldots q_{n-1} e_{n-1} q_n$, and let $G = (V_1 \cup \ldots \cup V_n, f_G, E)$ be the n-partite graph $\gamma(q_1 e_1 \ldots q_{n-1})$, given by induction. Let $G' = (U \cup U', f_{G'}, E')$ denote the bipartite graph of $q_{n-1} e_{n-1} q_n$. Then, we let $\gamma(\pi) = G \oplus G'$.

We define the *folded orbit graph* $\Gamma(\pi)$ for any path π that is not a cycle, as a bipartite graph on node set $\{1\} \times \mathcal{V}(\mathsf{first}(\pi)) \cup \{2\} \times \mathcal{V}(\mathsf{last}(\pi))$. There is an edge $\big((1, v), (2, w)\big)$ in $\Gamma(\pi)$ if, and only if there is a path from $(1, v)$ to (n, w) in $\gamma(\pi)$, where $n = |\pi| + 1$. Nodes are labelled by the regions they belong to. For any cycle π, we define $\Gamma(\pi)$ similarly on the node set $\mathcal{V}(\mathsf{first}(\pi))$. Thus $\Gamma(\pi)$ may contain cycles. We extend the operator $\oplus$ to folded orbit graphs. Figure 3.1 gives an example.

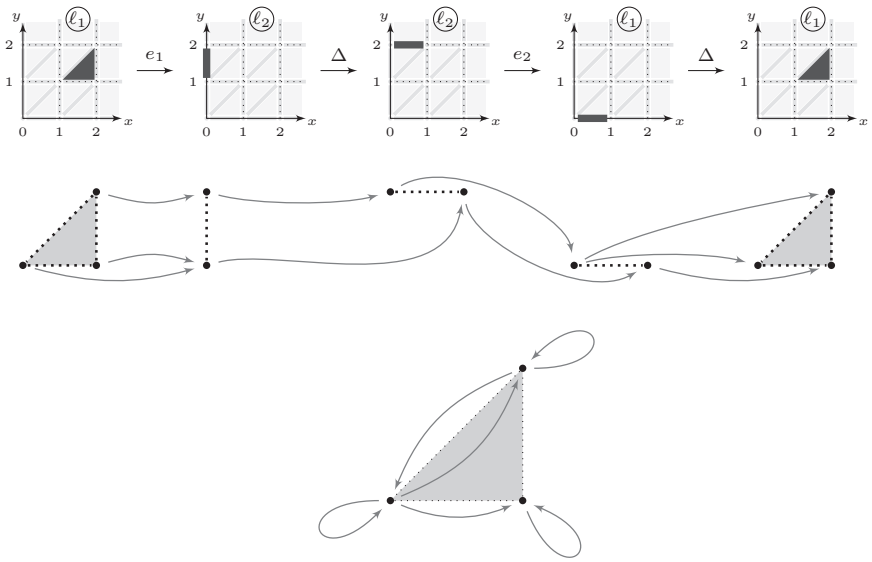


Figure 3.1: The orbit graph (at the middle) and folded orbit graph (at the bottom) of a cycle (on top) in the region automaton of $\mathcal{A}^2$ of Fig. 2.3.

3.3 Difference-Bound Matrices

Although the region automaton defined previously suffices to obtain the decidability of verification problems, exploring all regions is impractical due to their large number. In fact, the region abstraction (that is, the region automaton) is often too refined, in the sense that it distinguishes too many states. A more clever way of exploring the state space of timed automata is to use *zones*, which are arbitrary subsets of the state space definable by guards. Zones can be represented using *difference-bound matrices* (DBM). These matrices were used in [BM83, Dil90] in the context of state space exploration in timed systems.

Let $\mathcal{M}_n(K)$ denote the set of square matrices of size $n \times n$ with elements in K. Given a clock set $\mathcal{C}$, let $\mathcal{C}_0$ denote the set $\mathcal{C} \cup \{0\}$. A difference-bound matrix (DBM) is an element of $\mathcal{M}_{|\mathcal{C}_0|}(\mathbb{R} \times \{<, \leq\})$. We adopt the following notation: for any DBM M, we write $M = (\mathsf{M}, \prec^M)$, where M is the matrix made of the first components, with elements in $\mathbb{R} \cup \{\infty\}$, while $\prec^M$ is the matrix of the second components, with elements in $\{<, \leq\}$. A DBM M naturally represents a zone, denoted $[\![M]\!]$ (which we may also abusively write M), defined as the set of valuations $\nu \in \mathbb{R}^{\mathcal{C}}_{\geq 0}$ such that, for all $x, y \in \mathcal{C}_0$, it holds $\nu(x) - \nu(y) \prec^M_{x,y} \mathsf{M}_{x,y}$, where we set $\nu(0) = 0$. We will sometimes use the notation $[M]_{i,j}$ to write $M_{i,j}$ to ease reading.

We define a total order on $(\mathbb{R} \times \{<, \leq\}) \cup \{(\infty, <)\}$ as follows. We let

$$(a, \prec) \leq (b, \prec') \Leftrightarrow \begin{cases} a < b \\ \text{ or} \\ a = b \text{ and either } \prec = \prec' \text{ or } \prec' = \leq. \end{cases}$$

The addition is defined by $(a, \prec) + (b, \prec') = (a + b, \prec'')$, where $\prec'' = \leq$ if $\prec = \prec' = \leq$, and $<$ otherwise. We also let $(a, \prec) + b = (a + b, \prec)$.

For any DBM M, let $\mathsf{G}(M)$ denote the *constraint graph*, defined over nodes $\mathcal{C}_0$, which has an edge $(x, y) \in \mathcal{C}_0^2$ of weight $M_{x,y}$ if $M_{x,y} < (\infty, <)$. The *normalization* of M corresponds to assigning to each edge (x, y) the weight of the shortest path in $\mathsf{G}(M)$. This can be done in polynomial time using an all-pairs shortest path algorithm. We say that M is *normalized* when it is stable under normalization.

When exploring the state space of timed automata, several operations on DBMs are used. We recall here some of these that will be used in following chapters.

Definition 3.3.1. *Given normalized DBMs M and N, we let*

- $\mathsf{Pre}_{\mathsf{time}}(M)$ *is a DBM defining the set* $\{\nu \in \mathbb{R}^{\mathcal{C}}_{\geq 0} \mid \exists d \geq 0, \nu + d \in M\}$,
- $\mathsf{Post}_{\mathsf{time}}(M)$ *is a DBM defining the set* $\{\nu \in \mathbb{R}^{\mathcal{C}}_{\geq 0} \mid \exists d \geq 0, \nu - d \in M\}$,
- $M \cap N$ *is a DBM defining the set* $\{\nu \in \mathbb{R}^{\mathcal{C}}_{\geq 0} \mid \nu \in M, \nu \in N\}$,
- $\mathsf{Unreset}_R(M)$ *is a DBM defining the set* $\{\nu \in \mathbb{R}^{\mathcal{C}}_{\geq 0} \mid \nu[R \leftarrow 0] \in M\}$.

These operations can be computed in polynomial time. The computations are summarized in the following lemma; we refer to [BY04] for proofs.

Lemma 3.3.2. *Given normalized DBMs M and N,*

- $\mathsf{Pre}_{\mathsf{time}}(M)$ *is computed by setting the first row to* $(0, \leq)$ *and applying normalization.*

- $\mathsf{Post}_{time}(M)$ *is computed by setting the first column to* $(\infty, <)$.
- $M \cap N$ *is computed by assigning* $\min(M_{i,j}, N_{i,j})$ *to each component* (i, j)*, and applying normalization.*
- $\mathsf{Unreset}_R(M)$ *is computed by first intersecting* M *with the DBM defining* $\bigwedge_{x \in R}(x = 0)$*, then replacing each component* (i, j) *with* $i \in R$ *or* $j \in R$ *by* $(\infty, <)$*, and applying normalization.*

We note the following folklore result, see *e.g.* [HKSW11, Lemma 1].

Lemma 3.3.3. *For any normalized DBM* M*, and* $x, y \in \mathcal{C}_0$*, and* $\alpha \in (-M_{y,x}, M_{x,y})$*, there exists a valuation* ν *such that* $\nu(x) - \nu(y) = \alpha$.

3.4 Shrunk Difference-Bound Matrices

3.4.1 Motivation

To adapt the computation of the state space timed automata to perturbed semantics, we introduce an extension of the DBM data structure, called *shrunk difference-bound matrices* (*shrunk DBM* in short). The developments of this section were published in [SBM11, BMS12].

Let us first explain the idea on an example.

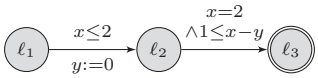


Figure 3.2: A Timed automaton.

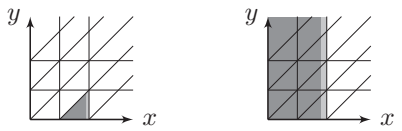


Figure 3.3: Winning states in ℓ_2 (left) and in ℓ_1 (right).

Consider the timed automaton of Fig. 3.2 under the excess-perturbation game semantics, with the objective of reaching ℓ_3. If there is no perturbation or lower bound on the delays between transitions (*i.e.*, $\delta = 0$), then the states from which Controller can reach location ℓ_3 can be computed backwards. One can reach ℓ_3 from location ℓ_2 and any state in the zone $X = (x \leq 2) \wedge (y \leq 1) \wedge (1 \leq x - y)$, shown by (the union of the light and dark) gray areas on Fig. 3.3 (left); this is the set of time-predecessors of the corresponding guard. The set of winning states from location ℓ_1 is the zone $Y = (x \leq 2)$, shown in Fig. 3.3 (right), which is simply the set of predecessors of X at ℓ_2. When $\delta > 0$ however, the set of winning states at ℓ_2 is a "shrinking" of X, shown by the dark gray area. If the value of the clock x is too close to 2 upon arrival in ℓ_2, Controller will fail to satisfy the guard $x = 2$ due to the lower bound δ on the delays. Thus, the winning states from ℓ_2 are described by $X \wedge (x \leq 2 - \delta)$. Then, this shrinking is backward propagated to ℓ_1: the winning states are $(x \leq 2 - 2\delta)$, where we "shrink" Y by 2δ in order to compensate for a possible perturbation.

An important observation here is that when $\delta > 0$ is small enough, so that both $X \wedge (x \leq 2 - \delta)$ and $(x \leq 2 - 2\delta)$ are non-empty, these sets precisely describe the winning states. Thus, we have a *uniform* description of the winning states for "all small enough $\delta > 0$".

3.4.2 Non-parameterized Shrunk DBMs

The idea behind shrunk DBMs is to represent zones whose facets are shrunk by infinitesimal amounts. Given a clock set $\mathcal{C}$, we define a *shrinking matrix (SM)*, as a nonnegative integer matrix of size $|\mathcal{C}_0| \times |\mathcal{C}_0|$. A shrunk DBM is written in the form $M - \delta P$, where M is a DBM, P a SM and δ is a fresh parameter. Let us explain how "shrunk zones", as in the above example, can be expressed using shrunk DBMs. Let M be a DBM, P a shrinking matrix, and $\delta > 0$. Suppose for example that for some i, $M_{i,0} = \alpha$, $M_{0,i} = \beta$, $P_{i,0} = k$ and $P_{0,i} = l$. Then, M defines the constraint "$-\beta \leq x \leq \alpha$" for some clock x, whereas $M - \delta P$ defines "$-\beta + l\delta \leq x \leq \alpha - k\delta$". Furthermore, if M represents a guard g, the shrunk guard $\langle g \rangle_{-\delta}$ (obtained by shrinking all atomic guards) can be represented by the shrunk DBM $M - \mathbf{1}\delta$, where matrix $\mathbf{1}$ has 0's on the diagonal and 1's everywhere else.

When manipulating shrunk DBMs, we will often be interested in the properties of shrunk DBMs $M - \delta P$ for "small enough $\delta > 0$", which means for all $\delta \in [0, \delta_0)$, for some $\delta_0 > 0$. We will use the notation (M, P), which means that we consider $M - \delta P$ for small enough $\delta > 0$. For instance, when we write $\mathsf{Pre}_{\mathsf{time}}((M, P)) = (N, Q)$, we mean that there exists $\delta_0 > 0$ such that $\mathsf{Pre}_{\mathsf{time}}(M - \delta P) = N - \delta Q$ holds for all $\delta \in [0, \delta_0)$. In all the operations, an upper bound on δ_0 will be computable.

In order to define operations on shrunk DBMs, we need to define a suitable algebra on the set of elements of the form $((\alpha, \prec), k)$ with $\alpha \in \mathbb{Q}$, $\prec \in \{<, \leq\}$ and $k \in \mathbb{N}$. We define addition as,

$$((\alpha, \prec), k) + ((\beta, \prec'), l) = ((\alpha + \beta, \prec''), k + l),$$

where $\prec'' = <$ if, and only if $\prec = <$ or $\prec' = <$. We also define the following order.

$$((\alpha, \prec), k) \preceq ((\beta, \prec'), l) \quad \Leftrightarrow \quad \begin{cases} \alpha < \beta \text{ or} \\ \alpha = \beta \text{ and } k > l, \text{ or} \\ \alpha = \beta \text{ and } k = l \text{ and } \prec' = < \ \Rightarrow \ \prec = <. \end{cases} \tag{3.1}$$

Minimum is then defined as follows.

$$\min(((\alpha, \prec), k), ((\beta, \prec'), l)) = \begin{cases} ((\alpha, \prec), k) & \text{if } ((\alpha, \prec), k) \preceq ((\beta, \prec'), l), \\ ((\beta, \prec'), l) & \text{otherwise.} \end{cases} \tag{3.2}$$

Notice that these operations make sense when we see the elements $((\alpha, \prec), k)$ as the right hand sides of inequalities of the form $x - y \prec \alpha - k\delta$ for small enough $\delta > 0$. For instance, the addition corresponds to summing two inequalities, while the minimum corresponds to the conjunction of two inequalities.

Thus, the order $\preceq$ and the minimum make sense for small enough $\delta > 0$. One can easily compute the maximum $\delta_0 > 0$ such that the desired inequality holds for all $\delta \in [0, \delta_0)$:

Lemma 3.4.1. *If $((\alpha, \prec), k) \preceq ((\beta, \prec'), l)$, then one can compute the greatest $\delta_0 > 0$ such that*

$$\forall x \in \mathbb{R}, x \prec \alpha - k\delta \quad \Rightarrow \quad x \prec' \beta - l\delta,$$

for all $\delta \in [0, \delta_0)$.

Proof. If $k = l$, then the inequality holds for all $\delta > 0$, so one can set $\delta_0 = \infty$. Otherwise one sets $\delta_0 = \frac{\alpha - \beta}{k - l}$. □

Normal forms As DBMs, shrunk DBMs have *normal forms*, which can be computed considering the shortest paths of the graph $\mathsf{G}(M)$ of a given DBM M. For a matrix A, we define the *A-weight of a path* $x_1, \ldots, x_n$ of $\mathsf{G}(M)$ as $\sum_{i=1}^{n-1} A_{x_i,x_{i+1}}$.

Definition 3.4.2. *Let M be a normalized DBM. For any $x, y \in \mathcal{C}_0$, we define $\Pi_{x,y}(\mathsf{G}(M))$ as the set of paths with least and finite M-weight from x to y in $\mathsf{G}(M)$.*

Notice that the shortest paths are defined with respect to weights in M and not M; we ignore the type of the inequalities. The *M-sign* of a path π, written $\mathsf{sign}_M(\pi)$ is $<$ if $\prec^M_{\pi_i,\pi_{i+1}} = <$ for some i, and $\leq$ otherwise.

Now, normalization consists in computing shortest paths in this algebra: Given a shrunk DBM (M, P) we define $\mathsf{norm}((M, P)) = (M', P')$ as follows: for each $(x, y) \in \mathcal{C}_0$, let $P'_{x,y}$ be the largest P-weight of the paths in $\Pi_{x,y}(\mathsf{G}(M))$. We let $M'_{x,y} = (\mathsf{M}_{x,y}, \prec^{M'}_{x,y})$ where $\prec^{M'}_{x,y} = <$ if some path of $\Pi_{x,y}(\mathsf{G}(M))$ of P-weight $P'_{x,y}$ has sign $<$, and $\leq$ otherwise.

Lemma 3.4.3. *Let M be a normalized non-empty DBM and P be a shrinking matrix. Then, (M, P) is non-empty if, and only if, for all $x \in \mathcal{C}_0$, there is no path in $\Pi_{x,x}(\mathsf{G}(M))$ with positive P-weight. Moreover, if (M, P) is not empty, then, writing $(M', P') = \mathsf{norm}((M, P))$, there exists $\delta_0 > 0$ such that $M' - \delta P'$ is normalized and defines the same set as $M - \delta P$ for all $\delta \in [0, \delta_0)$. The greatest such δ_0 is computable.*

Proof. First, note that $\mathsf{norm}((M, P))$ defines the same set as (M, P). In fact, we replace each component with the sum of other constraints of (M, P), which is already implied in (M, P).

Now, (M', P') satisfies the following, for all $x, y, z \in \mathcal{C}_0$,

$$\mathsf{M'}_{x,y} = \mathsf{M'}_{x,z} + \mathsf{M'}_{z,y} \Rightarrow (M'_{x,y}, P'_{x,y}) \preceq (M'_{x,z}, P'_{x,z}) + (M'_{z,y}, P'_{z,y}). \tag{3.3}$$

In fact, the condition means $(x, z, y) \in \Pi_{x,y}(\mathsf{G}(M))$, so, by definition of P', $P'_{x,y} \geq P'_{x,z} + P'_{z,y}$, where in case of equality, $\mathsf{sign}_{M'}(x, z, y) = <$ implies $\prec^{M'}_{x,y} = <$ by construction.

Now, let $\delta_0 > 0$ be small enough so that (3.3) hold for all $x, y, z \in \mathcal{C}_0$. Then, for all $\delta \in [0, \delta_0)$, (3.3) means $M'_{x,y} - \delta P'_{x,y} \preceq M'_{x,z} - \delta P'_{x,z} + M'_{z,y} - \delta P'_{z,y}$, which implies that the DBM $M' - \delta P'$ is normalized. Now, a normalized DBM is non-empty if, and only if all its diagonal entries are $(0, \leq)$. Therefore, (M, P) is empty whenever $\Pi_{x,x}(\mathsf{G}(M))$ has a path with positive P-weight, for some $x \in \mathcal{C}_0$. If any such path has P-weight equal to 0, then $\prec^M_{x,y} = \leq$ since M is non-empty.

To compute the greatest $\delta_0 > 0$ for which the statement holds, it suffices to apply Lemma 3.4.1 to (3.3) for all $x, y, z \in \mathcal{C}_0$. □

Remark 3.4.4. *Consider a normalized DBM M and any SM P. If $(M', P') = \mathsf{norm}((M, P))$, then $\mathsf{M}' = \mathsf{M}$, $M \subseteq M'$ and $P \leq P'$. This follows from the definition of normalization. In fact, normalization does not change the values M, and if $\prec^{M'}_{x,y} = <$, then $\Pi_{x,y}(\mathsf{G}(M))$ has a path with sign $<$, so $\prec^M_{x,y} = <$. That $P \leq P'$ follows from the fact that for all x, y, $P'_{x,y}$ is the greatest P-weight of the shortest paths from x to y, so it is at least $P_{x,y}$.*

Here is an example where the signs of the inequalities change due to normalization: Consider $(x \leq 1) \wedge (1 \leq y) \wedge (x - y < 1)$, which is normalized (seen as a DBM over two clocks). However the shrinking $(x \leq 1 - \delta) \wedge (1 \leq y) \wedge (x - y < 1)$, yields, after normalization $(x \leq 1 - \delta) \wedge (1 \leq y) \wedge (x - y \leq 1 - \delta)$, where the last inequality becomes large.

Operations For any interval $[a, b]$, we define the *shrinking operator* as

$$\mathsf{shrink}_{[a,b]}(Z) = \{v \mid v + [a, b] \subseteq Z\},$$

for any set Z. For a zone Z represented as a DBM, $\mathsf{shrink}_{[0,\delta]}(Z)$ is the DBM $Z - \delta \cdot \mathbb{1}_{\mathcal{C}\times\{0\}}$ and $\mathsf{shrink}_{[-\delta,\delta]}(Z)$ is the DBM $Z - \delta \cdot \mathbb{1}_{\mathcal{C}\times\{0\}\cup\{0\}\times\mathcal{C}}$, for any $\delta > 0$.

A crucial property of shrunk DBMs is that they are closed under standard operations used for exploring the state space of timed automata. The following lemma summarizes the computations.

Lemma 3.4.5. *Let M, M', N be normalized non-empty DBMs.*

1. *If $N = \mathsf{Pre}_{\mathsf{time}}(M)$, then for all SMs P, there exists a shrunk DBM (N', Q) such that $\mathsf{N} = \mathsf{N}'$, $N \subseteq N'$ and $(N', Q) = \mathsf{Pre}_{\mathsf{time}}((M, P))$. Moreover, Q is obtained from P, by setting $Q_{0,x} = 0$ for all $x \in \mathcal{C}$ and applying normalization.*

2. *If $N = M \cap O$, then for all SMs P and P', there exists a shrunk DBM (N', Q) such that $\mathsf{N}' = \mathsf{N}$, $N \subseteq N'$ and $(N', Q) = (M, P) \cap (O, P')$. Moreover, (N', Q) is given by setting*

$$(N'_{x,y}, Q_{x,y}) = \min((M_{x,y}, P_{x,y}), (O_{x,y}, P'_{x,y})),$$

   and applying normalization.

3. *If $N = \mathsf{Unreset}_R(M)$ for some $R \subseteq \mathcal{C}$, then for all SMs P, there exists a shrunk (N', Q) such that $\mathsf{N}' = \mathsf{N}$, $N \subseteq N'$ and $(N', Q) = \mathsf{Unreset}_R((M, P))$. To obtain (N', Q), first compute (N'', Q') such that $(N'', Q') = (M, P) \cap (R = 0)$ by previous case, then set all components (x, y) with $x \in R$ or $y \in R$, to $(N'', Q')_{x,y} = ((\infty, <), 0)$ and apply normalization.*

4. *If $N = \mathsf{Post}_{\mathsf{time}}(M)$, then for all SMs P, there exists a shrunk DBM (N, Q) such that $(N, Q) = \mathsf{Post}_{\mathsf{time}}((M, P))$. Q can be obtained from P by setting $Q_{x,0} = 0$ for all $x \in \mathcal{C}$.*

5. *If $(N, Q) = \mathsf{shrink}_{[-\delta,\delta]}((M, P))$, then $\mathsf{N} = \mathsf{M}$ and Q is obtained from P by incrementing all $P_{x,0}$ and $P_{0,x}$ for $x \in \mathcal{C}$. For shrink^+, one only increments $P_{x,0}$.*

One can compute the greatest $\delta_0 > 0$ such that the given property holds for all $\delta \in [0, \delta_0)$.

Proof. Let us assume that (M, P) is normalized. All operations are defined following corresponding ones for usual DBMs. For instance, to compute $\mathsf{Pre}_{\mathsf{time}}((M, P))$, one sets the first row to $((0, \leq), 0)$. In fact, for any $\delta > 0$, this is precisely the computation of $\mathsf{Pre}_{\mathsf{time}}$ on the DBM $M - \delta P$ (see Lemma 3.3.2). We then apply Lemma 3.4.3 to obtain a normalized shrunk DBM. The rest of the statement follows from Remark 3.4.4.

If (M, P) is not normalized, then we first normalization on (M, P) before applying the operations. If $N = f(M)$ denotes one of the equations in this lemma, because all operations are non-decreasing, if $(M', P') = \mathsf{norm}((M, P))$, then $N'' = f(M')$ satisfies $N \subseteq N''$, and moreover $\mathsf{N} = \mathsf{N}''$. The latter equality follows from $N \subseteq N''$ and the fact that any shortest path of N is also a shortest path of N'', though possibly with a different sign. Thus, by Remark 3.4.4, the lemma yields a shrunk DBM $(N', Q) = f((M', P')) = f((M, P))$ with $N \subseteq N'$ and $\mathsf{N} = \mathsf{N}'$, as desired. □

A function $f\colon \mathcal{M}_{|\mathcal{C}_0|}(\mathbb{Q}_\infty)^n \to \mathcal{M}_{|\mathcal{C}_0|}(\mathbb{Q}_\infty)$ (for some $n > 0$), is said *elementary* if it combines its arguments using operators $\mathsf{Pre}_{\mathsf{time}}$, $\mathsf{Post}_{\mathsf{time}}$, $\mathsf{Unreset}_R$, $\cap$, shrink and shrink^+.

Proposition 3.4.6. *Let $f\colon \mathcal{M}_{|\mathcal{C}_0|}(\mathbb{Q}_\infty)^n \to \mathcal{M}_{|\mathcal{C}_0|}(\mathbb{Q}_\infty)$ be an elementary function and $M_0, M_1, \ldots, M_n$ DBMs satisfying $M_0 = f(M_1, \ldots, M_n)$, and let $P_1, \ldots, P_n$ be SMs. Then, there exists a shrunk DBM (M', Q) with $\mathsf{M}' = \mathsf{M}$, $M \subseteq M'$ and $(M', Q) = f\big((N_1, P_1), \ldots, (N_k, P_k)\big)$. The shrunk DBM (M', Q) and the largest $\delta_0 > 0$ can be computed in polynomial time.*

Proposition 3.4.6 shows that shrinking matrices are propagated when usual operations are applied on DBMs. Using this, one can also compute an upper bound on δ, under which a given equation between shrunk DBMs hold. Figure 3.4 illustrates some of these operations. Notice also that the lemma always gives *normalized* shrunk DBMs (M', Q).

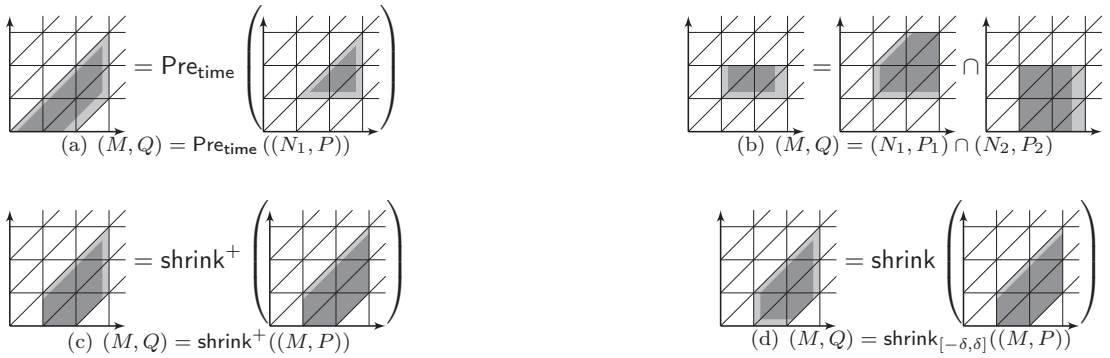


Figure 3.4: On the right of Fig. 3.4(a), the whole gray area represents a zone N_1, while the dark gray area is some shrinking (N_1, P). On the left, the dark area is the shrunk zone (M', Q) where $M = \mathsf{Pre}_{\mathsf{time}}(N_1)$, $\mathsf{M} = \mathsf{M}'$, and $M \subseteq M'$. Similarly, Fig. 3.4(b) to 3.4(d) illustrate the intersection of two shrunk zones and the shrinking of zones.

We define $\mathsf{Pre}_{\geq\delta}(M) = \mathsf{shrink}_{[0,\delta]}(\mathsf{Pre}_{\mathsf{time}}(M))$. It is easily observed that this is the set of states that can reach M after a delay of length at least δ: $\mathsf{shrink}_{[0,\delta]}(\mathsf{Pre}_{\mathsf{time}}(M)) = \{v \mid \exists d \geq \delta, v + d \in M\}$. We also define $\mathsf{Pre}_{>\delta}(M) = \{v \mid \exists d > \delta, v + d \in M\}$. The latter can be obtained from $\mathsf{Pre}_{\geq\delta}(M)$ by changing all upper bounds to strict inequalities.

Let us extend the computation of upper bounds on δ in Lemma 3.4.6 for an inclusion relation, which will be useful in Chapter 7.

Lemma 3.4.7. *Assume the equation $(M, P) \cap N \subseteq (N, Q)$ between normalized shrunk DBMs (M, P) and (N, Q). One can compute the greatest $\delta_0 > 0$, in polynomial time, such that $(M - \delta P) \cap N \subseteq N - \delta Q$ for all $\delta \in [0, \delta_0)$.*

Proof. Let $N' = M \cap N$. Let Q' be the SM given by Lemma 3.4.5 such that $(N', Q') = (M, P) \cap N$ and $\delta_0' > 0$ the corresponding bound on δ. Then, the inclusion is equivalent to $(N', Q') \subseteq (N, Q)$. Since both shrunk DBMs are normalized, holds if for all $x, y \in \mathcal{C}_0$, $((N'_{x,y}, \prec^{N'}_{x,y}), Q'_{x,y}) \prec ((N_{x,y}, \prec^{N}_{x,y}), Q_{x,y})$. Thus, if $\delta_1 > 0$ denotes the minimum of the upper bounds given by Lemma 3.4.1 for each x, y, we choose $\delta_0 = \min(\delta_0', \delta_1)$. □

Remark 3.4.8. *Although the results above allow one to compute the greatest $\delta_0 > 0$ satisfying all inequalities, one can obtain an easy upper bound, as follows. If the equation of Proposition 3.4.6 has a non-empty solution, then one can choose $\delta_0 = \frac{1}{3m}$, where m is the maximum of all components of the shrinking matrices that appear in the computations.*

Proof. Assume that for some valuation ν, $M_i - \delta Q_i$ satisfies the equation for small enough $\delta > 0$. We show that the equation is satisfied for all $\delta \in [0, \frac{1}{3m})$. To prove this, we need to show for all $\delta \in [0, \delta_0)$ that all shrunk DBMs are non-empty and satisfy the equations they are involved in.

Since all shrunk DBMs are non-empty (for small enough $\delta > 0$), all diagonals of the shrinking matrices are equal to 0. Thus, it suffices to show that all shrunk DBMs $M - \delta Q$ are normalized. Normalization condition requires

$$\forall i, j, k \in \mathcal{C}_0^3, \quad M_{i,j} - \delta Q_{i,j} \preceq M_{i,k} - \delta Q_{i,k} + M_{k,j} - \delta Q_{k,j},$$

for all $\delta \in [0, \delta_0)$. If $\mathsf{M}_{i,j} = \mathsf{M}_{i,k} + \mathsf{M}_{k,j}$, then we must have $Q_{i,j} \geq Q_{i,k} + Q_{k,j}$. So, for these components, the condition holds for all $\delta > 0$. If $\mathsf{M}_{i,j} < \mathsf{M}_{i,k} + \mathsf{M}_{k,j}$, then the condition holds if $\delta_0 < |\frac{\mathsf{M}_{i,k}+\mathsf{M}_{k,j}-\mathsf{M}_{i,j}}{Q_{i,k}+Q_{k,j}-Q_{i,j}}|$, but this is already the case since $\delta_0 \leq \frac{1}{3m}$ (the upper bound is infinity if the denominator is 0).

It remains to show that all equations that appear in the computations hold. For an equation of the form $(M, Q) = (N_1, R_1) \cap (N_2, R_2)$, we have either $(N_1)_{i,j} = (N_2)_{i,j}$ and $Q_{i,j} = \max((R_1)_{i,j}, (R_2)_{i,j})$, or for example $(N_1)_{i,j} < (N_2)_{i,j}$ and $(M_{i,j}, Q_{i,j}) = ((N_1)_{i,j}, (Q_1)_{i,j})$. In the former case, the equation holds for all $\delta > 0$. In the latter case, it holds for all $\delta < |\frac{(\mathsf{N}_2)_{i,j}-(\mathsf{N}_1)_{i,j}}{(Q_2)_{i,j}-(Q_1)_{i,j}}|$. This is already the case since $\delta < \frac{1}{3m}$. For equations of the form $(M, Q) = \mathsf{Pre}_{\mathsf{time}}((N, R))$, it suffices to choose δ small enough to ensure that normalization holds. For $(M, Q) = \mathsf{Unreset}_r((N, R))$, one only needs to ensure that the intersection with $r = 0$ and normalization holds. Both conditions were already shown to hold. □

How much can m grow? A rough estimation shows that it is at most exponential in the size of the equation. We haven't observed such a growth in practice; See Chapter 6 for experimental results.

3.4.3 Parameterized Shrunk DBMs

We now extend the shrunk DBM data structure to parameters. More precisely, we will consider shrinking matrices whose components are expressions with parameters. When studying shrinkability in Chapter 5, this will allow us to explore the state space of timed automata in a parameterized manner.

We only develop this data structure for closed sets. Since all operations we consider preserve closed sets, all DBMs in this section are assumed to use only closed inequalities. To simplify the presentation, we will drop the inequality types when writing the DBMs; a DBM M will be simply a matrix of real numbers.

We fix a tuple of parameters $\mathbf{k} = (k_i)_{i \in I}$, which will take nonnegative integer values. The *max-plus polynomials* over $\mathbf{k}$, denoted by $\mathcal{G}(\mathbf{k})$, are generated by the following grammar.

$$\phi ::= l \in \mathbb{N} \mid k_i,\ i \in I \mid \phi + \phi \mid \max(\phi, \phi).$$

A *parameter valuation* is a function that assigns a natural number to each parameter k_i. By a slight abuse of notation, we denote parameter valuations as $v \colon \mathbf{k} \to \mathbb{N}$. For any max-plus polynomial ϕ and parameter valuation $v \colon \mathbf{k} \to \mathbb{N}$, we denote by $\phi[v]$ the value of formula ϕ replacing each parameter k by $v(k)$. A *parameterized shrinking matrix (PSM)* is an element of $\mathcal{M}_{|\mathcal{C}_0|}(\mathcal{G}(\mathbf{k}))$. If P is a PSM and v is a parameter valuation, then $P[v]$ is the shrinking matrix defined by replacing each parameter k_i by $v(k_i)$, and evaluating the resulting expression.

As for non-parameterized shrunk DBMs, we need to provide an algebra in order to define our operations. We define symbolic operations on expressions of the form $\alpha - k \cdot \delta$, where $\alpha \in \mathbb{Q}$, k is an expression in $\mathcal{G}(\mathbf{k})$ (bound to take values in $\mathbb{N}$). As before, δ is seen as a positive parameter to be chosen small enough. More precisely, given two pairs (α, k) and (β, l) in $\mathbb{Q} \times \mathcal{G}(\mathbf{k})$, we define the addition $(\alpha, k) + (\beta, l) = (\alpha + \beta, k + l)$, where $(k + l)[v] = k[v] + l[v]$ for all $v \colon \mathbf{k} \to \mathbb{N}$. We also define the following relation:

$$(\alpha, k) \preceq (\beta, l) \quad \Leftrightarrow \quad \alpha < \beta \text{ or } (\alpha = \beta \text{ and } k[v] \geq l[v] \text{ for all } v \colon \mathbf{k} \to \mathbb{N}).$$

An important (but quite straightforward) property of this definition is the following. Notice that we are again interested in comparing (α, k) and (β, l) for small positive values of δ. Since all constraints are closed, we can compute the upper bounds δ_0 such that a given property holds for all $\delta \in [0, \delta_0]$.

Lemma 3.4.9. *For all (α, k) and (β, l) in $\mathbb{Q} \times \mathcal{G}(\mathbf{k})$, the following two statements are equivalent:*

- $(\alpha, k) \preceq (\beta, l)$
- *for all $v \colon \mathbf{k} \to \mathbb{N}$, there exists $\delta_0 > 0$ s.t. for all $0 \leq \delta \leq \delta_0$, it holds $\alpha - k[v] \cdot \delta \leq \beta - l[v] \cdot \delta$. Moreover, one can compute the greatest $\delta_0 > 0$ with this property.*

Proof. Assume that the second statement holds. In particular when $\delta = 0$, we get $\alpha \leq \beta$. Hence either $\alpha < \beta$, and we are done, or $\alpha = \beta$. In the latter case, we get that for all v, it holds $k[v] \cdot \delta \geq l[v] \cdot \delta$ for small positive values of δ. Hence for all v, $k[v] \geq l[v]$.

Conversely, we consider several cases:

- if $\alpha < \beta$: then for all $v \colon \mathbf{k} \to \mathbb{N}$,
  - either $k[v] = l[v]$ and any non-negative δ satisfies $\alpha - k[v] \cdot \delta \leq \beta - l[v] \cdot \delta$ (so that δ_0 can be any positive real);
  - or $|k[v] - l[v]| \geq 1$, in which case we let $\delta_0 = \frac{\beta - \alpha}{|k[v] - l[v]|}$. Then any nonnegative $\delta \leq \delta_0$ satisfies $|k[v] - l[v]| \cdot \delta \leq \beta - \alpha$. In particular, $(l[v] - k[v]) \cdot \delta \leq \beta - \alpha$, as required.
- if $\alpha = \beta$ and $k[v] \geq l[v]$ for all v: then for any non-negative δ, $k[v] \cdot \delta \geq l[v] \cdot \delta$, and the result follows.

□

Notice that $\preceq$ is not an ordering relation: indeed, $(\alpha, k) \preceq (\beta, l) \preceq (\alpha, k)$ implies $\alpha = \beta$ and $k[v] = l[v]$ for all v, but the latter does not imply equality of k and l (syntactically). In the sequel, we (silently) consider the quotient of $\mathcal{G}(\mathbf{k})$ by this equivalence relation, so that $\preceq$ is a partial order. To see that not all elements are comparable, consider $(1, k_1\delta)$ and $(1, (k_2 + k_3)\delta)$. For some parameter valuations, the former is smaller than the latter, and for some others the inverse is true.

We now define the *minimum* of two elements of $\mathbb{Q} \times \mathcal{G}(\mathbf{k})$:

$$\min((\alpha, k), (\beta, l)) = \begin{cases} (\alpha, k) & \text{if } \alpha < \beta, \\ (\beta, l) & \text{if } \beta < \alpha, \\ (\alpha, \max(k, l)) & \text{otherwise.} \end{cases}$$

This definition enjoys the following property:

Lemma 3.4.10. *For all* (α, k) *and* (β, l) *in* $\mathbb{Q} \times \mathcal{G}(\mathbf{k})$, $\min((\alpha, k), (\beta, l))$ *belongs to* $\mathbb{Q} \times \mathcal{G}(\mathbf{k})$ *and is the greatest lower bound of* (α, k) *and* (β, l) *in that set.*

Proof. The first statement is obvious, since $\max(k, l)$ belongs to $\mathcal{G}(\mathbf{k})$ as soon as k and l do. That $\min((\alpha, k), (\beta, l)) \preceq (\alpha, k)$ is easily obtained by considering three cases:

- if $\alpha < \beta$, then $\min((\alpha, k), (\beta, l)) = (\alpha, k)$, which implies our result;
- if $\beta < \alpha$, then $\min((\alpha, k), (\beta, l)) = (\beta, l)$, and $(\beta, l) \preceq (\alpha, k)$ since $\beta < \alpha$;
- if $\alpha = \beta$, then $\min((\alpha, k), (\beta, l)) = (\alpha, \max(k, l))$, and $(\alpha, \max(k, l)) \preceq (\alpha, k)$ since $(\max(k, l))[v] \geq k[v]$ for all v (by definition of max).

Symmetrically, $\min((\alpha, k), (\beta, l)) \preceq (\beta, l)$.

Now, consider any (γ, m) s.t. $(\gamma, m) \preceq (\alpha, k)$ and $(\gamma, m) \preceq (\beta, l)$. We show that then either $\min((\alpha, k), (\beta, l)) = (\gamma, m)$, or $\min((\alpha, k), (\beta, l)) \not\preceq (\gamma, m)$.

- When $\alpha < \beta$, assume $\min((\alpha, k), (\beta, l)) \preceq (\gamma, m)$. This means $(\alpha, k) \preceq (\gamma, m) \preceq (\alpha, k)$, which entails equality. The case where $\beta < \alpha$ is symmetric.
- When $\alpha = \beta$, assume again that $\min((\alpha, k), (\beta, l)) = (\alpha, \max(k, l)) \preceq (\gamma, m)$. This is easily seen to entail $\gamma = \alpha$, as we must also have $(\gamma, m) \preceq (\alpha, k)$. Moreover, for all v, it must hold $(\max(k, l))[v] \geq m[v]$, as well as $m[v] \geq k[v]$ and $m[v] \geq l[v]$. When $k[v] \geq l[v]$, we get $m[v] = k[v] = (\max(k, l))[v]$; when $l[v] > k[v]$, then we have $m[v] = l[v] = (\max(k, l))[v]$, so that finally $m = \max(k, l)$.

□

Normal forms We first show that shrunk DBMs also have a *normal form* in the following sense. Just like for DBMs, and non-parameterized shrunk DBMs, the normalization of a shrunk DBM (M, P) is obtained simply by computing the shortest path between all indices i and j, by interpreting (M, P) as the adjacency matrix of a directed graph. The algorithm for normalization is the Floyd-Warshall all-pairs shortest path algorithm applied to regular DBMs, but we apply it in our algebra over $\mathbb{Q} \times \mathcal{G}(\mathbf{k})$ where the sum, the order $\preceq$ and min are defined as above. We explicitly give the algorithm this time, in Algorithm 1, since the computation of shortest paths are now less obvious; they consist in building max-plus polynomials that encode these.

Algorithm 1 Normalization procedure for shrunk DBMs.

Given a shrunk DBM (M, P),
for $i = 0..n$ **do**
  for $j = 0..n$ **do**
    for $k = 0..n$ **do**
      $(M_{i,j}, P_{i,j}) \leftarrow \min\big((M_{i,j}, P_{i,j}), (M_{i,k}, P_{i,k}) + (M_{k,j}, P_{k,j})\big)$.
    end for
  end for
end for

Operations The following set of lemmas explains how elementary operations can be computed on shrunk DBMs. In particular, it shows how shrinking parameters (*i.e.*, the PSMs) can be propagated in a backward analysis while staying in the max-plus theory. In all the lemmas below, the resulting parameterized shrunk DBMs, and the greatest δ_0 with the stated property are computable in polynomial time (see also Proposition 3.4.15).

Lemma 3.4.11. *Let M be any DBM and P be a PSM. Then, there exists a PSM P' such that for all parameter valuations $v\colon \mathbf{k} \to \mathbb{N}$, there exists $\delta_0 > 0$ for which $\mathsf{norm}(M - \delta \cdot P[v]) = M' - \delta \cdot P'[v]$ for all $\delta \in [0, \delta_0]$, where $M' = \mathsf{norm}(M)$.*

Proof. We initialize the matrix (N, Q) as the matrix of $\mathcal{M}_{|\mathcal{C}_0|}(\mathbb{Q} \times \mathcal{G}(\mathbf{k}))$ whose element in cell (i, j) is $(M_{i,j}, P_{i,j})$. We first prove that each step of the normalization algorithm applied to $R = (N, Q)$ in $\mathcal{M}_{|\mathcal{C}_0|}(\mathbb{Q} \times \mathcal{G}(\mathbf{k}))$ yields a matrix in $\mathcal{M}_{|\mathcal{C}_0|}(\mathbb{Q} \times \mathcal{G}(\mathbf{k}))$. More precisely, given $R = (N, Q)$ and integers i, j and k less than or equal to C, we prove that the matrix R' obtained from R by replacing $R_{i,j}$ with $\min(R_{i,j}, R_{i,k} + R_{k,j})$ can be written as (N', Q') for some DBM N' and $Q' \in \mathcal{M}_{|\mathcal{C}_0|}(\mathcal{G}(\mathbf{k}))$.

Assume that $N_{i,j} \leq N_{i,k} + N_{k,j}$. If the inequality is strict, then $(N_{i,j}, Q_{i,j})$ equals $\min(R_{i,j}, R_{i,k} + R_{k,j})$. Otherwise $N_{i,j} = N_{i,k} + N_{k,j}$: then $R'_{i,j}$ is set to $(N_{i,j}, \max(Q_{i,j}, Q_{i,k} + Q_{k,j}))$, which satisfies our requirement. Assume now that $N_{i,k} + N_{k,j} < N_{i,j}$. In this case, $R'_{i,j} = R_{i,k} + R_{k,j}$, and the result follows from Lemma 3.4.10. The upper bound on δ, under which the inequalities hold can be computed using Lemma 3.4.10.

Let us write (M', P') the shrunk DBM returned by the normalization algorithm. By the definition of the minimum on pairs $(N_{i,j}, Q_{i,j})$, the algorithm applies normalization on the DBM N, so $M' = \mathsf{norm}(M)$. Pick any parameter valuation $v\colon \mathbf{k} \to \mathbb{N}$. We define a negative cycle of a shrunk DBM $(M, P[v])$ as $i_1, i_2, \ldots, i_k \in \mathcal{C}_0$, where $i_1 = i_k$, and $(M_{i_1,i_2}, P_{i_1,i_2}) + (M_{i_2,i_3}, P_{i_2,i_3}[v]) + \ldots + (M_{i_{k-1},i_k}, P_{i_{k-1},i_k}[v]) \prec (0, 0)$. if $(M, P[v])$ contains a negative cycle, then for any $\delta > 0$, $M - \delta P[v]$ is empty since this would imply the constraint $x - x \leq M'_{i_1,i_1} - \delta P'_{i_1,i_1}[v] < 0$, for some clock x. Otherwise, we have $\mathsf{norm}(M - \delta \cdot P[v]) = M' - \delta \cdot P'[v]$ for all small enough $\delta > 0$. In fact the operations performed on the parameterized expressions during the normalization algorithm correspond to the normalization algorithm applied on the DBM $M - \delta P$, by Lemma 3.4.10. Last, by applying Lemmas 3.4.9 and 3.4.10 to the (finitely many) cells of the matrix (M', P') we get $\delta_0 > 0$ such that such that for all $\delta \in [0, \delta_0]$ and for all indices i, j and k,

$$M_{i,j} - P'_{i,j}[v] \cdot \delta \leq M_{i,k} - P'_{i,k}[v] \cdot \delta + M_{k,j} - P'_{k,j}[v] \cdot \delta,$$

so that $M - P'[v] \cdot \delta$ is in normal form. Also, quite obviously, for any v and δ as above, the DBM obtained at each step of the normalization algorithm defines the same set of parameter valuations as the original DBM. □

Note that even if a DBM M is not empty, a shrunk DBM (M, P) can be empty. For instance, the shrunk DBM corresponding to constraints $1 + \delta \leq x \leq 1 - \delta$ is empty for all $\delta > 0$, although the set $[\![1 \leq x \leq 1]\!]$ is not. The emptiness of a shrunk DBM (M, P), *i.e.* whether $M - \delta P$ is empty for all $\delta > 0$, can be determined by first applying normalization, then checking whether all diagonal components are 0.

The intersection of two shrunk DBMs can also be written as a shrunk DBM:

Lemma 3.4.12. *Let M, N^1, N^2 be normalized DBMs such that $M = N^1 \cap N^2$. Then, for all PSMs P^1 and P^2, there exists a PSM P' such that for all parameter valuations $v\colon \mathbf{k} \to \mathbb{N}$, there exists $\delta_0 > 0$ for which $M - \delta \cdot P'[v] = (N^1 - \delta \cdot P^1[v]) \cap (N^2 - \delta \cdot P^2[v])$ for all $\delta \in [0, \delta_0]$.*

Proof. For all i, j, define $(N_{i,j}, P_{i,j}) = \min((N^1_{i,j}, P^1_{i,j}), (N^2_{i,j}, P^2_{i,j}))$. Then by definition, $N_{i,j} = \min(N^1_{i,j}, N^2_{i,j})$, so that $[\![N]\!] = [\![M]\!]$. Applying Lemma 3.4.9 (several times), for any parameter valuation v, there is a positive δ_0 for which $N_{i,j} - P_{i,j}[v] \cdot \delta = \min({N^1}_{i,j} - {P^1}_{i,j}[v] \cdot \delta, {N^2}_{i,j} - {P^2}_{i,j}[v] \cdot \delta)$, for all $0 \leq \delta < \delta_0$, so that the DBM $N - P[v] \cdot \delta$ corresponds to the intersection of $N^1 - P^2[v] \cdot \delta$ and $N^2 - P^2[v] \cdot \delta$.

To conclude, it suffices to apply Prop. 3.4.11 to turn the resulting shrunk DBM in normal form. □

We now describe how we compute the shrinking matrix for the *unreset* operation: given a DBM M and a set of clocks Z, $\mathsf{Unreset}_Z(M)$ is the (normal-form) DBM defining all clock parameter valuations v s.t. $v[Z := 0]$ belongs to M. It is obtained by intersecting with the DBM representing the set $Z = 0$, and removing contraints involving clocks in Z.

Lemma 3.4.13. *Let M and N be two normalized DBMs with $M = \mathsf{Unreset}_Z(N)$ for some $Z \subseteq \mathcal{C}$. Then, for any PSM P, there exists a PSM P' such that for all parameter valuations $v \colon \mathbf{k} \to \mathbb{N}$, there exists $\delta_0 > 0$ for which $M - P'[v] \cdot \delta = \mathsf{Unreset}_Z(N - P[v] \cdot \delta)$ for all $\delta \in [0, \delta_0]$.*

Proof. We first consider the intersection of the shrunk DBM (N, P) and the DBM representing the set $Z = 0$. From Prop. 3.4.12, this can be written as a shrunk DBM (N', P'), where N' is the normalized DBM obtained after intersecting N with the DBM for $Z = 0$. The second step consists in removing the constraints involving clocks of Z. This is achieved on DBMs by turning all the values in the corresponding columns to ∞; for shrunk DBMs, we change $(M_{i,j}, P_{i,j})$ into $(\infty, 0)$ whenever $i \in Z$ or $j \in Z$ except when $i \neq j$, the value of $P_{i,j}$ being actually not relevant in that case. This results in a shrunk DBM (M, Q) with the required properties.

Note that if (N, P) is empty, then so is (M, Q) since we did not change the diagonal, and that the components of the shrinking matrix can only increase during normalization. □

Finally, we compute the time-predecessor set of a DBM by dropping the lower-bound constraints on single clocks (*i.e.*, the first row of the DBM), while preserving all other constraints.

Lemma 3.4.14. *Let M and N be two normalized DBMs with $M = \mathsf{Pre}_{\mathsf{time}}(N)$. Then for any PSM P, there exists a PSM P' such that for all parameter valuations $v \colon \mathbf{k} \to \mathbb{N}$, there exists $\delta_0 > 0$ for which $M - \delta \cdot P'[v] = \mathsf{Pre}_{\mathsf{time}}(N - \delta \cdot P[v])$ for all $\delta \in [0, \delta_0]$.*

Proof. P' is obtained from P by also changing all the elements of the first row into 0 (except for the component $(0, 0)$), and applying normalization. Note that if (N, P) is empty, so is (M, Q), as in the previous lemma, since the diagonal of the shrinking matrix can only increase. □

By combining the previous lemmas, and Lemma 3.4.9, we immediately get the following proposition.

Proposition 3.4.15. *Let $f \colon \mathcal{M}_{|\mathcal{C}_0|}(\mathbb{Q}_\infty)^n \to \mathcal{M}_{|\mathcal{C}_0|}(\mathbb{Q}_\infty)$ be an elementary function and $M_0, M_1, \ldots, M_n$ DBMs satisfying $M_0 = f(M_1, \ldots, M_n)$. For any PSMs $P_1, \ldots, P_n$, one can compute a PSM P_0 s.t. for all $v \colon \mathbf{k} \to \mathbb{N}$, there exists a greatest (computable) $\delta_0 > 0$ such that*

$$(M_0, P_0[v]) = f\big((M_1, P_1[v]), \ldots, (M_n, P_n[v])\big),$$

for all $\delta \in [0, \delta_0]$. These computations can be carried out in polynomial time, and in particular P_0 has size polynomial in the size of $P_1, \ldots, P_n$ and f.

Observe that in the above equation, for any parameter valuation v, $(M, P'[v])$ is empty whenever some $(M_i, P_i[v])$ is empty.

Proposition 3.4.15 gives a polynomial-size PSM P' provided that we represent the max-plus polynomials by sharing subexpressions. For instance, we assume that if we have max-plus polynomials ϕ_1 and ϕ_2 in memory, then the expression $\max(\phi_1, \phi_2)$ is represented by a new node max() that point to ϕ_1 and ϕ_2. More precisely, these expressions can be given as max-plus graphs defined in Section 5.4.2. Then, each elementary operation adds a polynomial number of nodes (in the number of clocks), thus yielding a polynomial-size representation.

The bound δ_0 can be computed when successively applying Lemmas 3.4.11–3.4.14, using Lemma 3.4.9. In fact, such a choice of δ ensures that all shrunk DBMs encountered in the computations are non-empty and normalized or $\delta \in [0, \delta_0]$. By Remark 3.4.8, δ_0 can also be chosen as, roughly, the inverse of the maximal component of all SMs that appear in computations.

Our definition of PSM is different from parametric DBMs considered for instance in [HRSV01], since we use max-plus polynomials instead of linear expressions and only consider natural number valuations. A related parameterized extension of DBMs was used in [DDMR08] for the forward computation of the state space of enlarged timed automata. The emphasis in [DDMR08] was, however, on computing over-approximations on reachables states; (fixpoint) equations between sets were not studied.

Part II

Robustness Analysis

This part is devoted to the contributions on robustness analysis algorithms. In robustness analysis, the goal is to check whether a given system satisfies its specification when its semantics is perturbed. If this is the case, one is interested in synthesizing an upper bound on the magnitude of perturbations under which the specification is satisfied. We consider, in this part, two perturbation models, namely, enlargement and shrinking. Chapter 4 shows the decidability of the untimed language preservation under parameterized guard enlargements; while Chapter 5 shows the decidability of behavior preservation in terms of time-abstract simulation and non-blockingness under parameterized guard shrinkings. A software tool that can check the shrinkability of a given timed automaton, and some experimental results are presented in Chapter 6.

Chapter 4

Untimed Language Preservation

4.1 Introduction

Parameterized robust model-checking for timed automata, that is, deciding the existence of $\delta > 0$ for which a given property holds on the timed automaton enlarged by δ, has been shown to be decidable for several classes of properties. The problem has no complexity cost over the standard model-checking for timed automata: it was shown to be PSPACE-complete for safety [Pur00, DDMR08], and for ω-regular properties [BMR06, BMS11], and EXPSPACE-complete for a fragment of the metric temporal logic [BMR08]. In this chapter, we are interested in a stronger notion of robustness, asking for the preservation of the untimed language of the exact semantics in the enlarged timed automaton, for some value of the enlargement parameter $\delta > 0$. We call such timed automata *language robust*. In particular, if a timed automaton is language robust, then any (untimed) language based property (such as linear-time properties) proven for $[\![\mathcal{A}]\!]$ are preserved in $[\![\mathcal{A}_\delta]\!]$ for small enough δ.

Remember that, given a timed automaton $\mathcal{A}$, $L(\mathcal{A})$ denote the set of the untimed traces of the runs of $\mathcal{A}$. By definition of the enlargement, we have $L(\mathcal{A}_\delta) \subseteq L(\mathcal{A}_{\delta'})$ for any $\delta \leq \delta'$, and in particular $L(\mathcal{A}) \subseteq L(\mathcal{A}_\delta)$ for any $\delta \geq 0$. We are interested in the inverse inclusion, which does not always hold. In fact, one can show that in the automaton $\mathcal{A}^1$ of Figure 2.2, for all $\delta > 0$, all long enough words in $(\mathsf{gen} \cdot \mathsf{con})^* \cdot \mathsf{gen}^2$ belong to $L(\mathcal{A}^1_\delta)$ but not to $L(\mathcal{A}^1)$ (since location err is reachable in $[\![\mathcal{A}^1_\delta]\!]$ if, and only if $\delta > 0$).

Definition 4.1.1 (Language-robustness)**.** *A timed automaton $\mathcal{A}$ is* language-robust *if there exists $\delta > 0$ such that $L(\mathcal{A}) = L(\mathcal{A}_\delta)$.*

Informally, $\mathcal{A}$ is language-robust if $[\![\mathcal{A}_\delta]\!]$ has no extra behavior than $[\![\mathcal{A}]\!]$ for some $\delta > 0$, in terms of untimed language. Observe that whenever $L(\mathcal{A}_\delta) \subseteq L(\mathcal{A})$, we also have $L(\mathcal{A}'_\delta) \subseteq L(\mathcal{A})$ for any $0 < \delta' < \delta$. This is a desirable property, called "faster is better" [DDR05a, AT05], which means that once we prove the correctness of the system for some δ, it remains correct on any faster platform.

The main result of this chapter is that language robustness is in EXPSPACE. We also note a class for which it can be decided in PSPACE. The high complexity of the algorithm in the general case is not surprising, since deciding untimed language inclusion for timed automata is already EXPSPACE-complete [BGS12]. We do not have a lower bound result, but this algorithm conforms with other parameterized robust model-checking algorithms that have the same complexity as the corresponding exact cases.

In order to establish our algorithm, we revisit some results of [Pur00, DDMR08] and extend these to take into account the untimed languages (Section 4.4). Then, we prove a Ramsey-like combinatorial theorem on directed paths, which has an independent interest (Section 4.5). The proof of the main result (Section 4.6) combines these results.

The results of this chapter were published in [San11].

4.2 Restrictions on Timed Automata

Following [DDMR08, Pur00], we only consider closed timed automata and *rectangular* guards (that is, we do not have *diagonal* constraints such as $k \leq x - y \leq l$). We also assume that all clocks are bounded above by some constant M. Considering closed guards is natural in our setting, since we are interested in the behavior of the systems under positive enlargement. Assuming rectangular guards and bounded clocks is not restrictive in terms of expressiveness, but has an effect on the size of the models [BC05]. As in [DDMR08, Pur00, AMPS98], the only real restriction is the following. We consider timed automata where all clocks are reset at least once along any cycle of the region automaton; these are called *progress cycles*. A sufficient condition for a timed automaton to have only progress cycles is that any cycle of the underlying finite automaton resets all clocks at least once [AMPS98].

Although we prove our results for general timed automata with progress cycles, we also note a subclass for which we improve the complexity of the problem we study. We call a timed automaton *region-deterministic* if its timed-action region automaton is deterministic (that is, from all states of the region automaton, there is at most one outgoing edge per label)[1].

Note that we will mostly use region automata in the proofs of this chapter. Timed-action region automata will be useful only in the proof of Theorem 4.3.1

4.3 Main Result

Theorem 4.3.1. *Let $\mathcal{A}$ be any closed timed automaton with progress cycles, and W the size of its region automaton. Let $K = W$ if $\mathcal{A}$ is region-deterministic, and $K = 2^W$ otherwise, and fix any $N_0 \geq 15 \cdot W \cdot |\mathcal{C}|^2 \cdot 2^{(|\mathcal{C}|+1)^2} \cdot (K+1)^2$. Then, there exists $\delta > 0$ such that $L(\mathcal{A}_\delta) = L(\mathcal{A})$ if and only if $L(\mathcal{A}_{\frac{1}{N_0}}) = L(\mathcal{A})$.*

Our main result, that is, the decidability of language-robustness is a direct corollary of the previous theorem. In fact, $\mathcal{A}_{\frac{1}{N_0}}$ can be transformed into a (language-)equivalent integral automaton by multiplying all constants by N_0. We will denote by $\mathcal{R}(\mathcal{A}_{\frac{1}{N_0}})$ the region automaton of the corresponding integral timed automaton. We can then check whether $\mathcal{R}(\mathcal{A}_{\frac{1}{N_0}})$ and $\mathcal{R}(\mathcal{A})$ recognize the same untimed language. We obtain the following complexity results.

Corollary 4.3.2. *For region-deterministic timed automata with progress cycles, language robustness can be decided in* PSPACE. *For general timed automata with progress cycles, language robustness can be decided in* EXPSPACE.

Proof. Consider a region-deterministic timed automaton $\mathcal{A}$, and let $\mathcal{R}(\mathcal{A})$ denote its region automaton. Let $\mathcal{R}(\mathcal{A})^c$ denote the complement of $\mathcal{R}(\mathcal{A})$, which has thus the same size as $\mathcal{R}(\mathcal{A})$. One can decide whether $L(\mathcal{A}_{\frac{1}{N_0}}) \cap L(\mathcal{R}(\mathcal{A})^c) \neq \emptyset$ in polynomial space. In fact, the states of both $\mathcal{R}(\mathcal{A}_{\frac{1}{N_0}})$ and $\mathcal{R}(\mathcal{A})^c$ can be encoded in polynomial space (for $\frac{1}{N_0}$ given by the theorem for region-deterministic $\mathcal{A}$). Then, the usual non-deterministic procedure (see [AD94]) that guesses an accepting path in the product of these can be carried out in polynomial space.

[1] One could extend the definition of region-determinism by requiring that all states that satisfy the guards of edges with the same label to be untimed language-equivalent (that the same untimed language is recognized from those states). In fact, in this case, the region automaton can be made deterministic by leaving one (arbitrary) edge per label at each state.

For general timed automata, we describe a non-deterministic exponential space algorithm to decide $L(\mathcal{R}(\mathcal{A}_{\frac{1}{N_0}})) \not\subseteq L(\mathcal{R}(\mathcal{A}))$. Observe that $\mathcal{R}(\mathcal{A})$ can be complemented using the subset construction, and that each state in the complemented automaton has exponential size (since there are exponentially many regions). Let us call the deterministic complement automaton $\mathcal{R}(\mathcal{A})^c$. Thus, one can guess a path in $\mathcal{R}(\mathcal{A})^c$ in exponential space. The algorithm consists in guessing a path in $\mathcal{R}(\mathcal{A}_{\frac{1}{N_0}})$ while, in parallel, simulating the corresponding path in $\mathcal{R}(\mathcal{A})^c$. This can be done in exponential space since states of $\mathcal{R}(\mathcal{A}_{\frac{1}{N_0}})$ and $\mathcal{R}(\mathcal{A})^c$ can be represented and explored in exponential space, The algorithm accepts if the simulating set becomes empty, and otherwise rejects after a doubly exponential number of steps. In fact, $L(\mathcal{R}(\mathcal{A}_{\frac{1}{N_0}})) \not\subseteq L(\mathcal{R}(\mathcal{A}))$ is equivalent to $L(\mathcal{R}(\mathcal{A}_{\frac{1}{N_0}})) \cap L(\mathcal{R}(\mathcal{A})^c) \neq \emptyset$, and in this case, the intersection contains a word of size at most doubly exponential, since the product automaton has this size. □

In the rest of this chapter, we present the proof of Theorem 4.3.1. We start with a study of the properties of enlarged timed automata (Section 4.4), and some combinatorial results (Section 4.5), then give the proof (Section 4.6).

4.4 Timed Automata Under Enlargement

Let us fix a timed automaton $\mathcal{A}$ with $C > 0$ clocks. We start with the following result, which is a direct corollary of [San10, Lemma 3.14]. It states that for any $\delta > 0$, the trace of any run of $[\![\mathcal{A}_\delta]\!]$ of length $O(\lfloor \frac{1}{\delta} \rfloor)$ can be followed in $\mathcal{R}(\mathcal{A})$ too. An immediate implication is that if the length of the runs are fixed a priori, then a small enough enlargement has no effect on the behavior of timed automata (in terms of untimed language). Figure 4.1 illustrates the construction of the following lemma.

Lemma 4.4.1 ([San10]). *Fix any $n \in \mathbb{N}$ and $\delta > 0$ such that $\delta \leq \frac{1}{5nC^2}$. Let ρ be any run of $[\![\mathcal{A}_\delta]\!]$. Then, for all $1 \leq i_0 \leq |\rho|$, there exists a region, denoted by $H(\rho, i_0, n)$, included in $\mathsf{reg}(\mathsf{state}_{i_0}(\rho))$, such that for all regions $r \subseteq \overline{H(\rho, i_0, n)}$, there is a path π of $\mathcal{R}(\mathcal{A})$ over the trace $\mathsf{utrace}(\rho_{i_0 \dots \min(i_0+n, |\rho|)})$, with $\mathsf{state}_1(\pi) = r$ and $\mathsf{state}_j(\pi) \cap \overline{H(\rho, i_0 + j - 1, n)} \neq \emptyset$ for all $1 \leq j \leq |\pi|$.*

We are now interested in "long" or infinite runs. We saw that for some timed automaton $\mathcal{A}^1$ (*e.g.* Fig. 2.2) some regions that are not reachable in $[\![\mathcal{A}^1]\!]$ become entirely reachable in $[\![\mathcal{A}^1_\delta]\!]$, for any $\delta > 0$. An analysis of the behavior of $[\![\mathcal{A}^1_\delta]\!]$ shows that this is due to the accumulation of the "error" of δ along some cycles of the region automaton, as shown in Fig. 2.10. Puri gives a characterization of those cycles of $\mathcal{R}(\mathcal{A})$ which cause this behavior and obtain a decision procedure for safety properties [Pur00]; these proofs are revisited in [DDMR08]. In this section, we revisit the analysis of the cycles of $\mathcal{R}(\mathcal{A})$ under enlargement, and extend these results to a slightly more general setting. Roughly, we show that, the states that are reachable in $[\![\mathcal{A}_\delta]\!]$ by repeating a single cycle are also reachable by repeating particular sets of cycles in any order. Our proofs follow [DDMR08].

We say that two cycles π_1 and π_2 are *equivalent* if $\mathsf{first}(\pi_1) = \mathsf{first}(\pi_2)$ and $\Gamma(\pi_1) = \Gamma(\pi_2)$.

Let π be a cycle of $\mathcal{R}(\mathcal{A})$ starting at some state (ℓ, r). For any $k > 0$, we let

$$V_{\pi,k} = \{(\ell, v) \mid v \in \mathcal{V}(r), (v, v) \in \Gamma(\pi)^k\}.$$

This is the set of vertices $q \in \overline{\mathsf{first}(\pi)}$ for which there are runs following π^k that start and end at q. We define the convex hull of the union of these sets as $\mathsf{Lim}_\pi = \mathsf{convex\text{-}hull}(\bigcup_{k>0} V_{\pi,k})$. It is clear from the definition of Lim_π that $\mathsf{Lim}_\pi = \mathsf{Lim}_{\pi'}$ for any equivalent cycles π and π'.

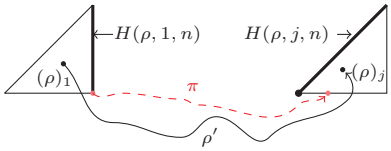


Figure 4.1: A run ρ of $[\![\mathcal{A}_\delta]\!]$. The leftmost triangle represents $\mathsf{reg}(\mathsf{state}_1(\rho))$, and the rightmost one $\mathsf{reg}(\mathsf{state}_j(\rho))$ (their corners, edges and interiors are subregions). By Lemma 4.4.1, there is a region $H(\rho, 1, n)$ such that starting from any region in $\overline{H(\rho, 1, n)}$, one can construct a path π of length n (the red dashed curve) such that $\overline{(\pi)_j}$ intersects $\overline{H(\rho, j, n)}$ for all j.

The main result of this section is the following lemma, which generalizes Theorem 23 of [DDMR08] (see also Lemma 7.10 in [Pur00]). We adopt the following notation: when we write $q \in (\ell, r)$, we mean that $q = (\ell, \nu)$ for some $\nu \in r$. We also write $\lambda(\ell, \nu) = (\ell, \lambda\nu)$ where $\lambda \in \mathbb{R}_{\geq 0}$, ν is a valuation and ℓ a location.

Lemma 4.4.2. *Let $\pi_1, \ldots, \pi_p$ be equivalent cycles of $\mathcal{R}(\mathcal{A})$ that start in state (ℓ, r), and consider any $\delta > 0$. Then, there exists $k > 0$ such that for any $q, q' \in (\ell, \overline{r})$, and any word $w \in \{\pi_1, \ldots, \pi_p\}^k$ there is a run in $[\![\mathcal{A}_\delta]\!]$ from q to q' on word w.*

To prove this lemma, we first show that Lim_{π_1} is backward and forward reachable in $[\![\mathcal{A}]\!]$, from any point of $\overline{r}$, by iterating at least C times any of the equivalent cycles in any order (Lemma 4.4.5), and that any pair of points in L_{π_1} can be connected by a run of $[\![\mathcal{A}_\delta]\!]$, again by iterating these cycles (Lemma 4.4.7).

A natural property of runs of timed automata is that convex combinations of two runs yield a run over the same word, as shown in the following lemma.

Lemma 4.4.3 ([DDMR08, Lemma 24], and [Pur00, Lemma 7.1])**.** *Let π be a path in $\mathcal{R}(\mathcal{A})$, and let ρ and ρ' be runs in $[\![\mathcal{A}]\!]$ that follow π. Then for all $\lambda \in [0, 1]$, there exists a run ρ'' of $[\![\mathcal{A}]\!]$ following π, such that $\mathsf{state}_i(\rho'') = \lambda\mathsf{state}_i(\rho) + (1 - \lambda)\mathsf{state}_i(\rho')$ for all $1 \leq i \leq n$.*

The following proposition provides a bound on the number of vertices of regions. It also implies that from each region, there is a finite number of cycles with pairwise distinct vertex maps. Remember that all clocks are bounded above by some constant, so we only need to consider bounded regions.

Lemma 4.4.4 ([DDMR08, Lemma 14])**.** *Any bounded region has at most $C + 1$ vertices. Any point $\nu \in \mathbb{R}^C_{\geq 0}$ is a convex combination of the vertices of $\mathsf{reg}(\nu)$.*

The following lemma states that, Lim_π is backward and forward reachable from any state of $\overline{\mathsf{first}(\pi)}$, by repeating at least C times cycles equivalent to π.

Lemma 4.4.5. *Let $\pi_1, \ldots, \pi_p$ be equivalent cycles of $\mathcal{R}(\mathcal{A})$, that all start in state (ℓ, r), and fix any $q \in (\ell, \overline{r})$. Then, for any $k \geq C$ and any path $w \in \{\pi_1, \ldots, \pi_p\}^k$, there exists $q_1, q_2 \in \mathsf{Lim}_{\pi_1}$ and runs ρ_1 and ρ_2 of $[\![\mathcal{A}]\!]$ that follow w, such that $\mathsf{first}(\rho_1) = q$ and $\mathsf{last}(\rho_1) = q_1$; and $\mathsf{first}(\rho_2) = q_2$ and $\mathsf{last}(\rho_2) = q$.*

Proof. We first prove the statement for $q = (\ell, v)$ with v a vertex. Remember that for all $v \in \mathcal{V}(r)$, there exists at least one $v' \in \mathcal{V}(r)$ such that $(v, v') \in \Gamma(\pi_1)$, and the number of vertices is at most $C + 1$ by Lemma 4.4.4, so by repeating C times any cycles among $\pi_1, \ldots, \pi_p$, we get a sequence

of vertices $v_1, \ldots, v_{C+1}$ such that $(v_i, v_{i+1}) \in \Gamma(\pi_1)$. But then, $v_i = v_j$ for some $i < j$, thus we have $v_i, v_{i+1}, \ldots, v_j \in \mathsf{Lim}_{\pi_1}$. Now, we can extend the sequence $v_1 \ldots v_j$ to the length k elements by repeating the cycle $v_i v_{i+1} \ldots v_j$. Clearly, this run can be constructed following any path w since π_i's are equivalent.

Consider now an arbitrary point q in $(\ell, \overline{r})$. By Lemma 4.4.4, q can be written as a convex combination of the vertices of r. Let $v_1, \ldots, v_m$ denote the vertices of r, and $\lambda_1, \ldots, \lambda_m \geq 0$ be such that $\lambda_1 + \ldots + \lambda_m = 1$ and $q = \lambda_1(\ell, v_1) + \ldots + \lambda_m(\ell, v_m)$. As we showed above, for any v_i, there is a run in $[\![\mathcal{A}]\!]$ that follows w, from v_i to some vertex $v_i' \in \mathsf{Lim}_{\pi_1}$. Lemma 4.4.3 yields the desired run. □

Lemma 4.4.6. *Let $\pi_1, \ldots, \pi_p$ be equivalent cycles in $\mathcal{R}(\mathcal{A})$. Then there exists $m > 0$ such that for all paths $w \in \{\pi_1, \ldots, \pi_p\}^m$, and for all $q \in \mathsf{Lim}_{\pi_1}$, there is a run ρ in $[\![\mathcal{A}]\!]$ from q to q, following w.*

Proof. By definition of Lim_{π_1}, any $z \in \mathsf{Lim}_{\pi_1}$ is a convex combination of a set of vertices v_i in Lim_{π_1}. But, for any vertex $v_i \in \mathsf{Lim}_{\pi_1}$, there exists $m_i > 0$ such that $(v_i, v_i) \in \nu(\pi_1)^{m_i}$. So, there exists $m > 0$ such that $(v_i, v_i) \in \nu(\pi_1)^m$ for all $1 \leq i \leq k$ (Consider for instance the least commun multiple). Now, the convex combination of these runs yield the desired run from q to q, by Lemma 4.4.3. □

The following lemma shows that any pair of states in L_π can be connected by a run in $[\![\mathcal{A}_\delta]\!]$.

Lemma 4.4.7 ([DDMR08, Lemma 29])**.** *Let π be a cycle of $\mathcal{R}(\mathcal{A})$ that starts in (ℓ, r) and let $q \in \mathsf{Lim}_\pi$. Then for any $\delta > 0$, and any $q' \in (\ell, \overline{r})$ such that $d_\infty(q, q') \leq \frac{\delta}{2}$, there is a run, in $[\![\mathcal{A}_\delta]\!]$, from q to q' following π.* □

Proof of Lemma 4.4.2. By Lemma 4.4.5, repeating at least C times the cycles $\pi_1, \ldots, \pi_p$ suffices to reach some point $q_1 \in \mathsf{Lim}_{\pi_1}$. The same lemma provides a point $q_2 \in \mathsf{Lim}_{\pi_1}$ which is backward reachable from q' by repeating C times any of these cycles. Because regions are convex, for any pair of points q_1, q_2 that belong to a same region, one can find points $q_1 = u_0, u_1, \ldots, u_N = q_2$ in $\overline{r}$ where $N = \lceil \frac{1}{\delta} \rceil$ such that $d_\infty(u_i, u_{i+1}) \leq \frac{\delta}{2}$ for all $0 \leq i \leq N-1$. Let $m > 0$ be the bound provided by Lemma 4.4.6. Now, q_2 can be reached from q_1, by a run over any word $\{\pi_1, \ldots, \pi_p\}^{mN}$ by Lemma 4.4.7 (applied N times to pairs (u_i, u_{i+1})). □

4.5 Some Combinatorial Tools

In this section, we prove a Ramsey-like theorem for colored directed paths, which gives a lower bound on the length of monochromatic subpaths contained in these (Subsection 4.5.1). This improves, by an exponential, the result provided by a direct application of Ramsey's theorem [Ram30]. In Subsection 4.5.2, we give a simple property on untimed finite automata accepting *ultimately universal* languages.

4.5.1 A Ramsey-like Theorem for Directed Paths

A *directed graph* is a pair $G = (V, E)$ where V is a finite set of *nodes* and $E \subseteq V \times V$, is the set of *edges*. A graph G is complete if for all $i, j \in V$ either $(i, j) \in E$ or $(j, i) \in E$. A graph is a *linearly-ordered complete graph*, if for some (strict) linear order $\prec$ on V, $(i, j) \in E$ iff $i \prec j$. The *degree* of a node v is $d(v) = |\{u \mid (v, u) \in E \text{ or } (u, v) \in E\}|$. Two nodes u and v are *connected* in the graph G if there exists a sequence $u = s_0, s_1, \ldots, s_k = v$ of nodes such that $(s_i, s_{i+1}) \in E$ or $(s_{i+1}, s_i) \in E$ for all $0 \leq i \leq k-1$. A graph is connected if all its nodes are connected. A *subgraph* of

$G = (V, E)$ is a graph (V', E') such that $V' \subseteq V$ and $E' \subseteq E$. A *connected component* is a maximal connected subgraph. A *directed path* of G is a sequence of nodes $v_1, \ldots, v_n$ such that $(v_i, v_{i+1}) \in E$ for all $1 \leq i \leq n-1$, and its *length* is n. A *r-coloring* of a graph $G = (V, E)$ is a function $E \to \{1, \ldots, r\}$ that associates to each edge a color from $\{1, \ldots, r\}$. A path is *monochromatic* if all its edges are assigned the same color.

Our result is based on the following theorem from [BGHS81].

Theorem 4.5.1 ([BGHS81])**.** *Let G be a connected directed graph over n nodes such that, for some h, every node v satisfies $d(v) \geq h$. Then G contains a directed path of length* $\min(n, h+1)$. □

The main result of this subsection is the following theorem.

Theorem 4.5.2. *Let $G = (V, E)$ be a linearly-ordered complete graph over n nodes given with an r-coloring of its edges. Then G contains a monochromatic directed path of length $\lfloor\sqrt{n/r - 2}\rfloor - 1$.*

Proof. Fix $h = \lfloor\sqrt{n/r - 2}\rfloor - 1$. For each $1 \leq i \leq r$, define subgraph G_i which contains exactly the edges colored by i. Then, for each G_i, define G'_i by removing any node v (and any edge containing v) whose degree in G_i is less than h. In G'_i, any node has either none or at least h edges of color i. Let G' be the union of all G'_i's. To define G', we remove at most $(h-1)rn$ edges ($h-1$ edges per color and node). Thus, G' has at least $\binom{n}{2} - (h-1)rn$ edges. Then, one of the G'_i's has at least $\binom{n}{2}/r - (h-1)n$ edges, say G'_{i_0}. We show that G'_{i_0} has a connected component C with at least $\sqrt{(n-1)/r - 2(h-1)}$ nodes. In fact, consider a graph with N nodes and M edges. Let K denote the number of nodes of the largest connected component. Since there are at most N connected components, each of them containing at most $\binom{K}{2}$ edges, we get that $M \leq N\binom{K}{2}$, which implies $\sqrt{2M/N} \leq K$. Applying this to G'_{i_0}, we get the desired connected component C. Now, by construction of G'_{i_0}, all nodes of C have degree at least h in C (in fact, by maximality, all edges of G'_{i_0} adjacent to the nodes of C are also included in C). Then, by Theorem 4.5.1, C contains a directed path of length at least $\min(\sqrt{(n-1)/r - 2(h-1)}, h)$. But the first term of the min is greater than or equal to h by the choice of h. So, there is a directed path of length at least $\lfloor\sqrt{n/r - 2}\rfloor - 1$ in G'_{i_0}, and this is a monochromatic path in G. □

4.5.2 Ultimately Universal Languages

Let Σ denote a finite alphabet. We define a *finite automaton* as a timed automaton without clocks. We call a regular language $L \subseteq \Sigma^*$ *ultimately universal* if there exists $k \geq 0$ such that $\bigcup_{l \geq k} \Sigma^l \subseteq L$.

Lemma 4.5.3. *Let L be an ultimately universal language recognized by a deterministic finite automaton with n locations. Then $\bigcup_{l \geq n-1} \Sigma^l \subseteq L$.*

Proof. Let A be a deterministic automaton that recognizes L. Suppose there is a word w of length at least n not recognized by A. Let l be the location reached in A by reading w. Since at least n locations are visited when reading w, we get that $w = tuv$ for some words t, u, v where u is non-empty, such that all words of tu^*v leads to location l in A. Because A is deterministic, no word of this set is recognized by A, contradicting the fact that L is ultimately universal. □

4.6 Proof of the theorem

Fix any timed automaton $\mathcal{A}$ with $C \geq 1$ clocks. Let W denote the number of regions of $\mathcal{A}$. Let $\mathcal{D}$ denote a deterministic finite automaton such that $L(\mathcal{D}) = L(\mathcal{R}(\mathcal{A})) = L(\mathcal{R}_{\text{ta}}(\mathcal{A}))$ (say, obtained

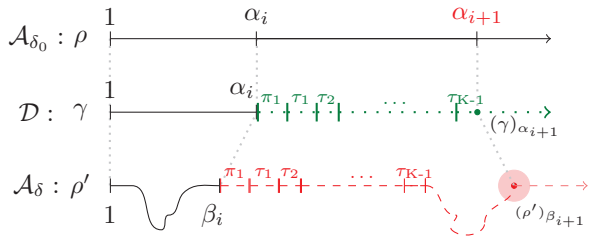


Figure 4.2: An induction step in the proof of Theorem 4.3.1. First, ρ' is extended following path $\pi_1\tau_1\tau_2\ldots$ (shown in red dashed line), and for any long enough repetition of cycles $\tau_1, \ldots, \tau_K$, $\overline{H(\rho, \alpha_{i+1}, n)}$ (shown in pink filled circle) can be reached. Then, γ is extended to $(\gamma)_{\alpha_{i+1}}$ (shown in green loosely dotted line).

by minimization), and let K denote the size of $\mathcal{D}$, which is at most 2^W ($K \leq W$ if $\mathcal{A}$ is region-deterministic). We let $M = 2^{(C+1)^2}\big((K+1)^2+2\big)$, $n \geq WM$ and fix any $\delta_0 \leq \frac{1}{5nC^2}$. The theorem states that if $L(\mathcal{A}) = L(\mathcal{A}_\delta)$ for some $\delta > 0$, then $L(\mathcal{A}) = L(\mathcal{A}_{\delta_0})$. The only interesting case is when $\delta < \delta_0$ since otherwise $L(\mathcal{A}) \subseteq L(\mathcal{A}_{\delta_0}) \subseteq L(\mathcal{A}_\delta)$ and the theorem follows. So let us suppose that $L(\mathcal{A}) = L(\mathcal{A}_\delta)$ for some $\delta < \delta_0$. Let ρ be a run of $[\![\mathcal{A}_{\delta_0}]\!]$. We will show that $\mathsf{utrace}(\rho) \in L(\mathcal{D}) = L(\mathcal{A})$, which will prove the theorem.

Lemma 4.6.1. *Any path π of $\mathcal{R}(\mathcal{A})$ of length at least n can be factorized as $\pi = \pi_1\tau_1\tau_2\ldots\tau_{K-1}\pi_2$ where π_1, π_2 are paths, and τ_i's are equivalent cycles.*

Proof. Since $n \geq W \cdot M$, by the Pigeon-hole principle, π contains a factor $t = t_1 \ldots t_M$ such that $\mathsf{first}(t_1) = \mathsf{first}(t_j)$ for all j. We apply Theorem 4.5.2 to get a further factorization of t. Consider a directed graph of the usual linear order $<$ over $\{1, \ldots, M\}$. To each edge (j, k) of the graph, where $j < k$, we assign as *color*, the vertex map $\Gamma(t_j t_{j+1} \ldots t_k)$. The number of colors is then bounded by $2^{(C+1)^2}$. Applying Theorem 4.5.2, we get that t contains a factor $\tau_1 \ldots \tau_{K-1}$, where $\tau_1 = t_{j_1} t_{j_1+1} \ldots t_{j_2}, \tau_2 = t_{j_2} t_{j_2+1} \ldots t_{j_3}, \ldots, \tau_{K-1} = t_{j_{K-1}} \ldots t_{j_K}$, for some $j_1 < j_2 < \ldots < j_K$, such that $\Gamma(\tau_1) = \Gamma(\tau_j)$ for all $1 \leq j \leq K$. □

Lemma 4.6.2. *Let $\pi = \pi_1\tau_1\tau_2\ldots\tau_{K-1}\pi_2$ be a path of $\mathcal{R}(\mathcal{A})$ where π_1 and π_2 are paths and τ_i's are equivalent cycles. Then, there exists $k_0 > 0$ such that for all $q \in \overline{\mathsf{first}(\pi)}$, $q' \in \overline{\mathsf{last}(\tau_{K-1})}$, $k \geq k_0$, and any word $w \in \mathsf{utrace}(\pi_1) \cdot (\mathsf{utrace}(\tau_1) + \ldots + \mathsf{utrace}(\tau_{K-1}))^k$, there is a run ρ' of $[\![\mathcal{A}_\delta]\!]$ over w with $\mathsf{first}(\rho) = q$ and $\mathsf{last}(\rho) = q'$.*

Proof. There is a run in $[\![\mathcal{A}]\!]$, from q to some $q_1 \in \overline{\mathsf{first}(\tau_1)}$ following π_1 since π_1 is a path of the region automaton of $\mathcal{A}$. This is also a run of $[\![\mathcal{A}_\delta]\!]$. The rest follows from Lemma 4.4.7 since $\mathsf{first}(\tau_1) = \mathsf{last}(\tau_{K-1})$. □

We are now ready to prove our main theorem. The reader may follow the proof in Figure 4.2.

Proof of Theorem 4.3.1. Consider ρ and the constants as defined above, and notice that $\delta_0 \leq \frac{1}{N_0}$. Let $H(\rho, i, n)$ be the regions given by Lemma 4.4.1 for all $i \geq 0$. We will inductively construct the desired run γ of $\mathcal{D}$ with $\mathsf{utrace}(\gamma) = \mathsf{utrace}(\rho)$. At step i of the induction, we will define $\gamma_{\alpha_i \ldots \alpha_{i+1}}$ for some increasing sequence $(\alpha_i)_{i\geq 0}$ with $\alpha_0 = 1$. When constructing γ, we will also construct an auxiliary run ρ' of $[\![\mathcal{A}_\delta]\!]$ in parallel, defining $\rho'_{\beta_i \ldots \beta_{i+1}}$ at each step i, for some increasing sequence $(\beta_i)_{i\geq 0}$ with

$\beta_0 = 1$, and ensuring that $\mathsf{state}_{\beta_i}(\rho') \in \overline{H(\rho, \alpha_i, n)}$ and $L([\![\mathcal{A}_\delta]\!], \mathsf{state}_{\beta_i}(\rho')) \subseteq L(\mathcal{D}, \mathsf{state}_{\alpha_i}(\gamma))$ for all $i \geq 0$.

– For $i = 0$, since ρ is an initialized run, we have $\mathsf{state}_1(\rho) = (l_0, \mathbf{0})$ so $H(\rho, 1, n) = \mathsf{reg}(l_0, \mathbf{0})$. We have $\alpha_0 = \beta_0 = 1$, $\mathsf{state}_1(\rho') = (l_0, \mathbf{0})$ and $\mathsf{state}_1(\gamma)$ is the initial state of $\mathcal{D}$. We have $L([\![\mathcal{A}_\delta]\!], \rho'_1) \subseteq L(\mathcal{D}, \gamma_1)$ by hypothesis.

– For any $i \geq 1$, suppose by induction that γ is defined between indices 1 and α_i and that $\rho'_{\beta_i} \in \overline{H(\rho, \alpha_i, n)}$. We will choose $\alpha_{i+1} > \alpha_i$ and $\beta_{i+1} > \beta_i$, and first define $\rho'_{\beta_i \dots \beta_{i+1}}$ such that $\mathsf{state}_{\beta_{i+1}}(\rho') \in \overline{H(\rho, \alpha_{i+1}, n)}$, then define $\gamma_{\alpha_i \dots \alpha_{i+1}}$. Let π be the path of $\mathcal{R}_{\mathrm{ta}}(\mathcal{A})$ which starts at $\mathsf{reg}(\mathsf{state}_{\beta_i}(\rho'))$, given by Lemma 4.4.1 for the run $\rho_{\alpha_i \dots |\rho|}$. If ρ is finite and $|\rho| - \alpha_i \leq n$, then $\mathcal{D}$ has a run from γ_{α_i} on word $\mathsf{utrace}(\pi)$ (since $L([\![\mathcal{A}_\delta]\!], \rho'_{\alpha_i}) \subseteq L(\mathcal{D}, \gamma_{\alpha_i})$) and we are done. Suppose now that ρ is either infinite, or $|\rho| - \alpha_i > n$. Then $|\pi| = n$, and by Lemma 4.6.1, π can be decomposed into $\pi = \pi_1 \tau_1 \dots \tau_{K-1} \pi_2$ where τ_i's are equivalent cycles. We let $\alpha_{i+1} > \alpha_i$ such that $\mathsf{last}(\tau_{K-1})$ is the $(\alpha_{i+1} - \alpha_i)$-th state of π. $[\![\mathcal{A}_\delta]\!]$ has a run from $\mathsf{state}_{\beta_i}(\rho')$ to some $z \in \mathsf{first}(\tau_1)$ following π_1 (in fact, $\mathsf{state}_{\beta_i}(\rho') \in \mathsf{first}(\pi_1)$). By construction of π, there exists $z' \in \overline{H(\rho, \alpha_{i+1}, n)} \cap \overline{\mathsf{last}(\tau_{K-1})} \neq \emptyset$, and by Lemma 4.6.2, for any $k \geq k_0$ and any word $w \in (\mathsf{utrace}(\tau_1) + \ldots + \mathsf{utrace}(\tau_{K-1}))^k$, there is a run, in $[\![\mathcal{A}_\delta]\!]$, from z to z' following trace w. Let $\rho''(w)$ denote the run thus constructed from $\mathsf{state}_{\beta_i}(\rho')$ to z' on $\mathsf{utrace}(\pi_1) \cdot w$. We let β_{i+1} s.t. $\rho'_{\beta_i \dots \beta_{i+1}}(w) = \rho''(w)$ for an arbitrary w.

Now, $\mathcal{D}$ has a run from $\mathsf{state}_{\alpha_i}(\gamma)$ to some state q_0 over trace $\mathsf{utrace}(\pi_1)$ because $L([\![\mathcal{A}_\delta]\!], \mathsf{state}_{\beta_i}(\rho')) \subseteq L(\mathcal{D}, \mathsf{state}_{\alpha_i}(\gamma))$. Let $\mathcal{D}'$ denote the finite untimed automaton obtained from $\mathcal{D}$ by designating q_0 as the initial state, and all states q_f such that $L([\![\mathcal{A}_\delta]\!], z') \subseteq L(\mathcal{D}, q_f)$ for some $w \in (\mathsf{utrace}(\tau_1) + \ldots + \mathsf{utrace}(\tau_{K-1}))^k$, $k \geq k_0$, as final states. There is at least one final state because $L(\mathcal{D}) = L(\mathcal{A}_\delta)$ and $\mathcal{D}$ is deterministic. Let there be an edge in $\mathcal{D}'$ with label $\mathsf{utrace}(\tau_i)$ from state q to q' whenever there is a path in $\mathcal{D}$ from q to q' over word $\mathsf{utrace}(\tau_i)$. Observe that $\mathcal{D}'$ is still deterministic. Since $\rho''(w)$ is defined for any $w \in (\mathsf{utrace}(\tau_1) + \ldots + \mathsf{utrace}(\tau_{K-1}))^k$, $k \geq k_0$, $\mathcal{D}'$, defined over alphabet $\{\mathsf{utrace}(\tau_1), \dots, \mathsf{utrace}(\tau_{K-1})\}$, is ultimately universal. But then, by Lemma 4.5.3, $\mathcal{D}'$ accepts any word in $\{\mathsf{utrace}(\tau_1), \dots, \mathsf{utrace}(\tau_{K-1})\}^{K-1}$, and in particular $\mathsf{utrace}(\tau_1 \dots \tau_{K-1})$. Therefore, there is a run in $\mathcal{D}$ from $\mathsf{state}_{\alpha_i}(\gamma)$ to some state $\mathsf{state}_{\alpha_{i+1}}(\gamma)$ following $\mathsf{utrace}(\tau_1 \dots \tau_{K-1})$, which satisfies $L([\![\mathcal{A}_\delta]\!], \mathsf{state}_{\beta_{i+1}}(\rho')) \subseteq L([\![\mathcal{A}]\!], \mathsf{state}_{\alpha_{i+1}}(\gamma))$. □

4.7 Another simple but expensive algorithm

In [BMS11], we gave a PSPACE algorithm for parameterized robust model-checking of timed automata against Büchi properties. The algorithm consists in model-checking $\mathcal{A}_{\delta_0}$ for some fixed δ_0 depending on $\mathcal{A}$. This can be used to deduce a 2EXPSPACE algorithm for language robustness.

The main result of [BMS11] is the following.

Lemma 4.7.1 ([BMS11]). *For any a timed automaton $\mathcal{A}$, and state-based Büchi property B, there exists $\delta > 0$ such that $\mathcal{A}_\delta$ satisfies B, if, and only if $\mathcal{A}_{\delta_0}$ satisfies B, where $\delta_0 = O(2^{-|\mathcal{A}|})$.*

We show how this lemma can be used to derive a 2EXPSPACE algorithm for deciding language robustness. Consider the deterministic finite automaton $\mathcal{F}$, obtained from $\mathcal{R}_{\mathrm{ta}}(\mathcal{A})$ by the subset construction. Since all states of $\mathcal{R}_{\mathrm{ta}}(\mathcal{A})$ are accepting (the language $L(\mathcal{A})$ is prefix-closed by definition), only the emptyset is accepting in $\mathcal{F}$, which is the complement of the subset construction. Note that this accepting state is a sink state; let us denote it $\perp$. We consider the timed automaton $\mathcal{B} = \mathcal{A} \times \mathcal{F}$. Let B denote the set of all states of $\mathcal{B}$ but those having $\perp$ as the second component. Let $\delta_0 = O(2^{-|\mathcal{B}|})$. Then, $\mathcal{B}_\delta$ satisfies B for some $\delta > 0$ if, and only if $\mathcal{B}_{\delta_0}$ does so. But satisfying B

means that all runs of $\mathcal{B}_\delta$ visit infinitely often states of B, hence do not visit $\bot$. This is equivalent to saying that $L(\mathcal{A}_\delta) \subseteq L(\mathcal{A})$, hence $\mathcal{A}$ is language robust.

Now, $\mathcal{F}$ can have doubly exponential size, so according to the above lemma, $1/\delta_0$ is triply exponentially large. This suggests a triply exponential-time algorithm. But one can improve the complexity as follows. Instead of explicitly computing the product $\mathcal{B}$, one can explore $\mathcal{A}_{\delta_0}$ and $\mathcal{F}$ on-the-fly, in doubly exponential space. In fact, regions of $\mathcal{A}_{\delta_0}$ can be represented and explored in doubly exponential space, while a state of $\mathcal{F}$ only requires exponential state (since it is a subset of states of $\mathcal{R}_{\text{ta}}(\mathcal{A})$). Then, it suffices to guess a lasso, of length at most triply exponential, visiting the states of B in these two automata in parallel.

4.8 Conclusion

We presented an EXPSPACE algorithm for deciding language robustness in timed automata, requiring the untimed language to be preserved under some enlargement. This is a strong notion, implying that for language-robust timed automata, all untimed trace-based properties proven for the exact semantics hold under enlargement. In particular, for such timed automata, one does not need to apply robust model-checking algorithms for untimed specifications.

A drawback of our algorithm is its high complexity. Although untimed language inclusion in timed automata is also EXPSPACE-complete [BGS12], this does not imply a lower bound for language robustness. Whether the problem is EXPSPACE-hard remains open. One hope to reduce complexity would be to consider the problem of *(bi)simulation robustness*, that is, checking whether a given timed automaton is similar or bisimilar to its enlargement, for some enlargement parameter. The complexity might be lower, as it is already the case for timed automata and finite-state systems (see *e.g.* [BGS12]).

Chapter 5

Shrinkability

5.1 Introduction

In this chapter, we study robustness in timed automata against guard shrinkings. The robustness notion we consider requires timed automata to preserve some time-abstract behaviors, and not to become blocking, when their guards are shrunk by a parameterized amount.

Shrinking the guards corresponds to disallowing time delays that are too close to the boundaries of the guards. Some timed automata are vulnerable to such restrictions, and do loose some time-abstract behaviors. We will see in the next section examples of timed automata that cannot repeat infinitely some of their cycles under shrinkings. Similar phenomena were reported in [CHR02, ACS10].

Our goal is to develop algorithms to decide whether all guards can be shrunk –by possibly different amounts, so that the resulting timed automaton can still time-abstract simulate the original automaton, and is non-blocking. By this robustness check, one ensures that the behavior of the automaton does not depend on exact timings, more precisely, on its ability to take the transitions on the boundaries of the guards. A shrinkable timed automaton preserves all its behaviors when, for instance, task execution times are shorter than the worst-case, and waiting times are longer than the best-case. One can also detect unrealistic runs, including Zeno runs (Section 5.2.2). Shrinkability analysis complements the robustness approach based on guard enlargement considered in Chapter 4.

The results of this chapter were published in [SBM11].

5.2 Robustness and Shrinkability

5.2.1 Shrinkability

Formally, a *shrinking* of a timed automaton $\mathcal{A}$ is $\mathcal{A}_{-\mathbf{k}\delta}$, where $\mathbf{k} \in \mathbb{N}^I_{>0}$ and $\delta > 0$, I denoting the set of all atomic clock constraints of $\mathcal{A}$. We are interested in deciding the existence of shrinking parameters $\mathbf{k}$ and $\delta > 0$, and in their computation, for which the shrunk timed automaton is non-blocking, or able to time-abstract simulate a given automaton, or both.

The formal definition of *non-blocking-shrinkability* is the following.

Definition 5.2.1. *Let $\mathcal{A}$ be a timed automaton, and I the set of its atomic guards. $\mathcal{A}$ is* non-blocking-shrinkable *if there exists $\mathbf{k} \in \mathbb{N}^I_{>0}$ and $\delta_0 \in \mathbb{Q}_{>0}$ such that for all $\delta \in [0, \delta_0]$, $[\![\mathcal{A}_{-\mathbf{k}\delta}]\!]$ is non-blocking.*

We now define *shrinkability*. Ideally, we would like the shrunk timed automaton to be able to time-abstract simulate the original timed automaton, which would mean that all time-abstract behaviours present in the original timed automaton are preserved. But we give a more general definition, which allows us to define shrinkings that contain *some* part of the time-abstract behaviour of the original automaton. Given a timed automaton $\mathcal{A}$, and some finite automaton $\mathcal{F}$ such that

$[\![\mathcal{F}]\!] \sqsubseteq_{\text{t.a.}} [\![\mathcal{A}]\!]$, $\mathcal{A}$ is said to be *simulation-shrinkable with respect to* $\mathcal{F}$, if some shrinking $\mathcal{A}_{-\mathbf{k}\delta}$ simulates $\mathcal{F}$. Notice that $\mathcal{F}$ can be chosen as the region automaton of $\mathcal{A}$ [AD94] or a smaller bisimulation quotient [TY01], in which case a shrinking is required to time-abstract simulate $\mathcal{A}$ entirely. Otherwise, simulation-shrinkability guarantees that the shrunk automaton contains some relevant behaviour, given as $\mathcal{F}$, of the original timed automaton.

We need the following notation. Given TTSs $\mathcal{S}$ and $\mathcal{T}$, and a state s of $\mathcal{S}$, we denote by $\text{ta-sim}_{\mathcal{T}}(s)$ the set of states of $\mathcal{T}$ that time-abstract simulates the state s of $\mathcal{S}$. The formal definition of simulation-shrinkability is the following.

Definition 5.2.2. *Let $\mathcal{A}$ be a timed automaton, and I the set of its atomic guards. Consider any finite automaton $\mathcal{F}$ such that $[\![\mathcal{F}]\!] \sqsubseteq_{t.a.} [\![\mathcal{A}]\!]$. $\mathcal{A}$ is* simulation-shrinkable w.r.t. *$\mathcal{F}$ if there exists $\mathbf{k} \in \mathbb{N}^I_{>0}$ and $\delta_0 \in \mathbb{Q}_{>0}$ such that for all $\delta \in [0, \delta_0]$,*

$$[\![\mathcal{F}]\!] \sqsubseteq_{t.a.} [\![\mathcal{A}_{-\mathbf{k}\delta}]\!]$$

with the following additional requirement: for each state f of $\mathcal{F}$, there exists a guard g and $\mathbf{h} \in \mathbb{N}^{|\mathcal{C}_0|^2}$ such that for all $\delta \in [0, \delta_0]$, $\text{ta-sim}_{[\![\mathcal{A}_{-\mathbf{k}\delta}]\!]}(f)$ equals $[\![\langle g \rangle_{-\mathbf{h}\delta}]\!]$.

We say that a timed automaton is *strongly shrinkable w.r.t.* $\mathcal{F}$ if it has a shrinking witnessing both its non-blocking-shrinkability and its simulation-shrinkability w.r.t. $\mathcal{F}$. The above $\mathbf{k}$ and δ_0 are called the *shrinking parameters* of $\mathcal{A}$.

We now comment on the above definitions. We define shrinkability "for all $\delta \in [0, \delta_0]$", so if an automaton is shrinkable, we require it to remain correct when imprecisions are reduced, that is when δ is chosen smaller. In fact, the shrunk automaton can be seen as an underapproximation of the initial automaton, and we would like to be able to obtain arbitrarily close correct approximations by only adjusting δ. Notice also that when a timed automaton is simulation-shrinkable, then we require that for all small enough δ, each simulator set can be expressed as shrinkings $\langle g \rangle_{-\mathbf{h}\delta}$ where $\mathbf{h}$ is the same for all δ (that is, parameters $\mathbf{h}$ are *uniform*). If this is the case, then when we change slightly the parameter $\delta > 0$, the simulator sets also change slightly. Moreover, simulator sets have a uniform expression, where δ is only a parameter. When one uses the simulator sets to control the original system, this property yields a uniform representation for the constraints to add to the system. Thus, the controlled system can still be represented as a timed automaton, where the guards contain the parameter δ. Note however that we do not know whether there exist timed automata that are shrinkable but violate the technical requirement of Definition 5.2.2.

Example 5.2.3. *We illustrate shrinkability on the timed automaton $\mathcal{A}$ of Fig. 5.1. This timed automaton is shrinkable both for non-blockingness and for simulation. One can check that $\mathcal{A}_{-\mathbf{k}\delta}$, shown in Fig. 5.2 is non-blocking, and can time-abstract simulate $\mathcal{A}$ for all $\delta \in [0, \frac{1}{6}]$. One can also see that the simulator sets are uniform. For instance, the set of states of $\mathcal{A}_{-\mathbf{k}\delta}$ that simulate the initial state of $\mathcal{A}$ is $[\![\delta \leq x \leq 3 - \delta \wedge y \leq 3 - 2\delta \wedge \delta \leq x - y \leq 2 - 2\delta]\!]$ for all $\delta \in [0, \frac{1}{6}]$. The computation of these parameters, and that of the simulator sets of this example are explained in Example 5.3.1.*

5.2.2 Shrinking as a Remedy to Unrealistic Behaviour

Shrinkability also excludes *unrealistic* timing constraints, such as those requiring Zeno behaviours. In fact, for any timed automaton $\mathcal{A}$, consider the automaton $\mathcal{A}'$ obtained from $\mathcal{A}$ by adding a new clock u, the constraint $u \geq 0$ and the reset $u := 0$ at every edge. Clearly, $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic.

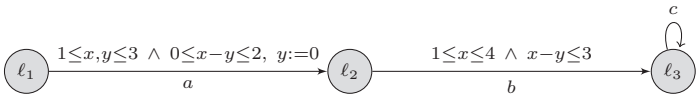


Figure 5.1: Timed automaton $\mathcal{A}$. Atomic clock constraints of the form $y < \infty$, $0 \leq y$ or $-\infty < x - y$ are omitted.

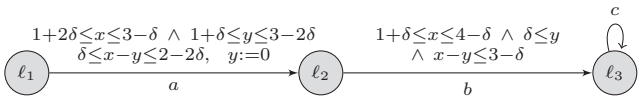


Figure 5.2: A shrunk timed automaton $\mathcal{A}_{-\mathbf{k}\delta}$ of timed automaton $\mathcal{A}$ defined in Fig. 5.1.

If automaton $\mathcal{A}'$ is, say, simulation-shrinkable, then $\mathcal{A}$ does not need Zeno strategies to satisfy the properties proven for the exact semantics and preserved by time-abstract similarity. In fact, each $u \geq 0$ is shrunk to some $u \geq \delta_i$ with $\delta_i > 0$, so time diverges in any infinite run.

But unrealistic timing constraints are not limited to Zeno behaviours. The automaton in Fig. 5.3 provides an example of a timed automaton which is non-blocking for $\delta_1 = \delta_2 = \delta_3 = 0$, and lets the time diverge but it becomes blocking whenever $\delta_2 > 0$ or $\delta_3 > 0$, so it is not shrinkable.

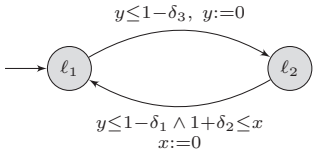


Figure 5.3: A shrunk timed automaton that is blocking whenever $\delta_2 > 0$ or $\delta_3 > 0$. To see this, consider any infinite execution and let $d_1, d_3, \ldots$ denote the delays at location l_1, and $d_2, d_4, \ldots$ those at l_2. One can show that $1 \leq d_{2i-1} + d_{2i}$ for all $i \geq 1$, which means that time diverges, but also $\delta_2 + \delta_3 \leq d_{2i+2} - d_{2i}$ and $d_{2i} \leq 1$. The latter means that the sequence $(d_{2i})_i$ increases at least by $\delta_2 + \delta_3$ at each step and is bounded above by 1, which is possible only when $\delta_2 = \delta_3 = 0$. Note that even if $\delta_2 = \delta_3 = 0$, non-blockingness requires consecutive delays to be equal, which is not realistic for digital systems.

A similar example appears in [CHR02] with equality constraints. The above example shows that such phenomena can occur even without the use of equality.

5.2.3 Decidability of Shrinkability

The main result of this chapter is the decidability of the shrinkability problems. A timed automaton is non-blocking if its TTS is. A timed automaton with *distinct labels* is a timed automaton in which all edges have distinct labels.

Theorem 5.2.4. *The following results hold:*

- *For closed non-blocking timed automata, non-blocking-shrinkability is in* PSPACE*, and in* NP *if the number of outgoing transitions from each location is bounded.*
- *For closed timed automata $\mathcal{A}$ with distinct labels, simulation-shrinkability w.r.t. any $\mathcal{F}$ is decidable in pseudo-polynomial time in the sizes of $\mathcal{A}$ and $\mathcal{F}$.*
- *For closed non-blocking timed automata with distinct labels, strong shrinkability w.r.t. $\mathcal{F}$ is decidable in time exponential in $\mathcal{A}$, and polynomial in $\mathcal{F}$, more precisely, in time $O(2^{|\mathcal{A}|} + p(|\mathcal{F}|, |\mathcal{A}|, M))$, where M is the largest constant of $\mathcal{A}$, and p some polynomial.*

In each case, the largest $\delta_0 > 0$ satisfying the property can be computed.

Moreover, we will show that when a given timed automaton is shrinkable, the least shrinking parameters can be computed (in a sense defined in Section 5.5). We assume distinct labels for simulation-shrinkability, mainly for technical reasons. Nevertheless, it is also meaningful for our purpose. In fact, we compare $\mathcal{A}$ with its shrinking and require these to have approximately the same behaviour. Checking simulation between these systems under this assumption not only requires "observational" equivalence but also a structural one. In fact, if a shrinking simulates $\mathcal{F}$, it should do so by following exactly the same edges as $\mathcal{A}$ does when simulating $\mathcal{F}$. Thus, we require the shrunk automaton and the initial automaton to have the same internal behaviour.

In the rest of this chapter, we present the proof of this result. We will use parameterized shrunk DBMs, presented in Chapter 3, give tools for solving fixpoint equations between parameterized shrunk DBMs by reduction to fixpoint equations in the max-plus algebra. We then explain how these results can be used to solve the shrinkability problems, by expressing them as fixpoint equations.

5.3 Equations on shrunk DBMs

Let us give an insight into the use of operations on shrunk DBMs to treat shrinkability problems. The following example explains the computation of the shrinking parameters on the timed automata of Fig. 5.1 and 5.2.

Example 5.3.1. *We consider the timed automaton of Fig. 5.1, with $g_1 = 1 \leq x, y \leq 3 \wedge 0 \leq x - y \leq 3$ the guard of the edge from ℓ_1 to ℓ_2, $R_1 = \{y\}$ its reset set, and $g_2 = 1 \leq x \leq 4 \wedge x - y \leq 3$ the guard of the edge from ℓ_2 to ℓ_3. We shrink the guard g_1 into*

$$g_1' = 1 + k_1\delta \leq x \leq 3 - k_2\delta \wedge 1 + k_3\delta \leq y \leq 3 - k_4\delta \wedge k_5\delta \leq x - y \leq 2 - k_6\delta,$$

and g_2 into

$$g_2' = 1 + k_7\delta \leq x \leq 4 - k_8\delta \wedge k_9\delta \leq y \wedge x - y \leq 3 - k_{10}\delta.$$

Assume that we are looking for a valuation of $\mathbf{k} = (k_i)_{1 \leq i \leq 10}$ in $\mathbb{N}_{>0}$ for which the resulting shrunk automaton $\mathcal{A}_{-\mathbf{k}\delta}$ witnesses the simulation-shrinkability, i.e. it can time-abstract simulate $\mathcal{A}$ for small enough $\delta > 0$. According to our definition of shrinkability, the simulator sets of $\mathcal{A}_{-\mathbf{k}\delta}$ must be shrinkings of the simulator sets of $\mathcal{A}$. Let us concentrate on three interesting simulation classes: all states $\ell_3 \times \mathbb{R}^{\mathcal{C}}_{\geq 0}$ are simulation-equivalent and can be extended to an infinite run, the set of states $X = [\![x \leq 4 \wedge 0 \leq x - y \leq 3]\!]$ at location ℓ_2 are those that can go to ℓ_3 by a b action, and the set of states $Y = [\![x, y \leq 3 \wedge 0 \leq x - y \leq 3]\!]$ at location ℓ_1 can go to X by an a action. One can see that X is precisely the time-predecessors of g_2, that is

$$X = \mathit{Pre}_{\mathit{time}}(g_2) \tag{5.1}$$

Further,

$$Y = \mathit{Pre}_{\mathit{time}}\,(g_1 \cap \mathit{Unreset}_y(X)) \tag{5.2}$$

expresses the fact that some point of $\ell_2 \times [\![X]\!]$ can be reached in one step starting from $\ell_1 \times [\![Y]\!]$, and this defines Y. Now, we use these equations to compute the simulator sets when guards are shrunk. Let X' denote the shrunk DBM describing time-predecessors of g'_2, as given by Lemma 3.4.14. We have $X' = [\![x \leq 4 - k_8\delta \wedge x - y \leq 3 - k_{10}\delta]\!]$, and $\ell_2 \times X'$ is indeed the set of states of $\mathcal{A}_{-\mathbf{k}\delta}$ that can simulate the states $\ell_2 \times X$ of $\mathcal{A}$, for any given valuation and small enough $\delta > 0$. Let us now compute Y', the simulator set in $\mathcal{A}_{-\mathbf{k}\delta}$ of the states Y of $\mathcal{A}$, applying Lemmas 3.4.11–3.4.14 to the equation relating Y to X. In the figure below, the union of dark gray and light gray areas illustrate this equation for $\delta = 0$, while the dark gray areas illustrate the equation between shrunk *zones, i.e. between X' and Y'. We have:*

$$\begin{aligned}
&\mathit{Unreset}_y(X') && = [\![x \leq 3 - k_{10}\delta]\!],\\
&[\![g'_1]\!] \cap \mathit{Unreset}_y(X') && = [\![1 + \max(k_1, k_3 + k_5)\delta \leq x \leq 3 - \max(k_2, k_{10})\delta\\
&&& \wedge 1 + k_3\delta \leq y \leq 3 - \max(k_4, k_5 + \max(k_2, k_{10}))\delta\\
&&& \wedge k_5\delta \leq x - y \leq 2 - \max(k_6, k_3 + \max(k_2, k_{10}))\delta]\!],\\
&Y' = \mathit{Pre}_{\mathit{time}}\,([\![g'_1]\!] \cap \mathit{Unreset}_y(X')) && = [\![k_5\delta \leq x \leq 3 - \max(k_2, k_{10})\delta\\
&&& \wedge y \leq 3 - \max(k_4, k_5 + \max(k_2, k_{10}))\delta\\
&&& \wedge k_5\delta \leq x - y \leq 2 - \max(k_6, k_3 + \max(k_2, k_{10}))\delta]\!].
\end{aligned}$$

The calculations are illustrated in Fig. 5.3.1. We now have at hand both parameterized expressions for the simulator sets X' and Y', given parameterized shrunk guards g'_1 and g'_2. It remains to choose a valuation, and check that X' and Y' are non-empty for small enough $\delta > 0$. We choose the valuation that sets $k_1 = k_4 = k_6 = 2$ and other parameters to 1. Note here that some parameters are set to 2 so that the shrunk guards are normalized.[1] *We get that under this valuation, $X' = [\![x \leq 4 - \delta \wedge x - y \leq 3 - \delta]\!]$ and $Y' = [\![\delta \leq x \leq 3 - \delta \wedge y \leq 3 - 2\delta \wedge \delta \leq x - y \leq 2 - 2\delta]\!]$. These sets and the guards are non-empty, and all equations above hold for all $\delta \in [0, \frac{1}{6}]$. This bound can be derived by looking at each application of Lemma 3.4.11–3.4.14 in our computations. Hence, we obtained a shrunk timed automaton $\mathcal{A}_{-\mathbf{k}\delta}$ that can time-abstract simulate $\mathcal{A}$, and expressions for the simulator sets parameterized by δ.* [2]

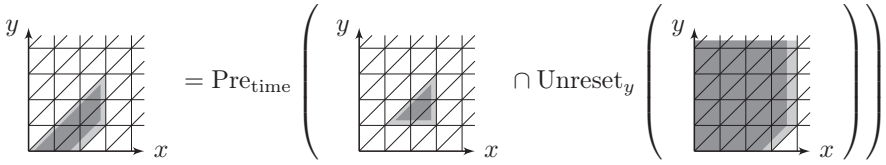


Figure 5.4: The calculations of Example 5.3.1. The light and dark gray areas combined describe the equation $Y = \mathsf{Pre}_{\mathsf{time}}\,(g_1 \cap \mathsf{Unreset}_y(X))$, while the dark areas describe the shrunk version: $Y' = \mathsf{Pre}_{\mathsf{time}}\,(g'_1 \cap \mathsf{Unreset}_y(X'))$.

In Example 5.3.1, we guessed a valuation $\mathbf{k}$ that witnessed the shrinkability of the considered timed automaton. The fact that the timed automaton did not contain cycles simplified the constraints on this valuation. The aim of the next section is to express systematically all constraints on parameters

[1] We could instead apply normalization to g'_1 and g'_2 and set all parameters to 1.

[2] Note that the initial state $\ell_1 \times \mathbf{0}$ is not included in X' unless we set $k_5 = 0$. This can be tested easily and we let the user decide whether simulation is considered to hold or not in this case.

k induced by the given equations, even in presence of cyclic dependencies, and provide an algorithm to compute a valuation satisfying these constraints.

We consider fixpoint equations on DBMs and study whether by "shrinking" a given solution, one can still satisfy the equation. Our goal is to generalize the arguments of Example 5.3.1 where we were able to compute a shrinking of the solutions to Equations (5.1) and (5.2). We consider fixpoint equations of the following form.

$$M_i = f_i(M_1, \ldots, M_n, M_{n+1}, \ldots, M_{n+n'}), \quad \forall 1 \leq i \leq n, \tag{5.3}$$

where $M_1, \ldots, M_{n+n'}$ are unknown normalized DBMs and f_i's are elementary functions. Notice that $M_{n+1}, \ldots, M_{n+n'}$ are *unconstrained*, *i.e.* they do not appear in the left hand side of the equation. Let us write $m = n + n'$. We assume we are given a solution $(M_i)_{1 \leq i \leq m}$ to Equation (5.3), and we are interested in *shrunk solutions* defined as follows.

Definition 5.3.2. *Consider a solution $(M_i)_{1 \leq i \leq m}$ of* (5.3). *A* shrunk solution of (5.3) w.r.t. $(M_i)_{1 \leq i \leq m}$ *is a triple*

$$\big((M_i)_{1 \leq i \leq m}, (Q_i)_{1 \leq i \leq m}, \delta_0\big),$$

where $\delta_0 > 0$ and Q_i's are shrinking matrices such that for all $\delta \in [0, \delta_0]$, $(M_i - \delta Q_i)_{1 \leq i \leq m}$ is a solution of (5.3). *A shrunk solution is called the* greatest shrunk solution *if $(Q_i)_{1 \leq i \leq m}$ are the least shrinking matrices that define a shrunk solution w.r.t. $(M_i)_{1 \leq i \leq m}$.*

Notice that we define shrunk solutions with shrinking matrices, not with parameterized ones. Here, the "least shrinking matrices" refer to the order on $\mathbb{N}^k$ defined by $\mathbf{a} \leq \mathbf{b}$ if, and only if $a_i \leq b_i$ for all $1 \leq i \leq k$. Clearly, the sets X' and Y' we computed in Example 5.3.1 are a shrunk solution (and in fact the least one) with respect to the solution (X, Y). A non-empty shrunk solution is a shrunk solution whose all shrunk DBMs are non-empty.

We now show how one can reduce the problem of finding shrunk solutions to that of solving simpler fixpoint equations in the max-plus algebra. For a non-empty solution $(M_i)_{1 \leq i \leq m}$ of (5.3), consider parameterized shrinking matrices P_i, for all $1 \leq i \leq m$, where each cell of each P_i is a unique parameter in **k**. By Proposition 3.4.15, there exist matrices $(\phi_i)_{1 \leq i \leq n}$ of max-plus polynomials such that

$$\big(M_i, \phi_i(P_1, \ldots, P_m)[v]\big) = f_i\big((M_1, P_1[v]), \ldots, (M_m, P_m[v])\big),$$

for all $1 \leq i \leq n$, and any parameter valuation v. Here, $\phi_i(P_1, \ldots, P_m)$ denotes a matrix whose components are max-plus polynomials that combine the components of matrices P_j. The above equation suggests that we study the following fixpoint equation on PSMs P_i's:

$$\begin{aligned} &P_i = \phi_i(P_1, \ldots, P_m), \quad \forall 1 \leq i \leq n, \\ &[P_i]_{j,j} = 0, \quad \forall 1 \leq i \leq m, 1 \leq j \leq |\mathcal{C}_0|. \end{aligned} \tag{5.4}$$

If some parameter valuation v satisfies (5.4), then if we denote by $Q_i = P_i[v]$ for all i, we get that $\big((M_i)_{1 \leq i \leq m}, (Q_i)_{1 \leq i \leq m}, \delta_0\big)$ is a shrunk solution of (5.3), for some $\delta_0 > 0$. In fact, the first line means that the fixpoint equation holds (by Prop. 3.4.15), and the second line means that this is a non-empty shrunk solution. Note that requiring the diagonals to be zero is sufficient since we assume all shrunk DBMs to be normalized. The converse also holds: the shrinking matrices of any shrunk solution of (5.3) satisfies (5.4). By Prop. 3.4.15, the size of the equation (5.4) is polynomial in the size of (5.3).

Note that we will be often interested in solutions of (5.4) where some parameters are positive. These parameters will, for instance, correspond to the shrinking of the guards. In order to enforce positive values to some parameters, one can augment Equation (5.4) with the following constraint, for any matrix cell $[P_i]_{j,k}$ that is to be positive:

$$[P_i]_{j,k} = \max(1, [P_i]_{j,k}). \tag{5.5}$$

The choice of the positive parameters depend on the application. For instance, in Section 5.5, we will shrink all the atomic constraints of given timed automata by positive amounts.

Let us define the *positivity function* as $\mathcal{P}(x) = 0$ if $x = 0$ and $\mathcal{P}(x) = \infty$ otherwise. We extend $\mathcal{P}(\cdot)$ to matrices, by componentwise application. The correspondance between shrunk solutions and max-plus equations is formally stated in the following lemma.

Lemma 5.3.3. *Consider any non-empty solution $(M_i)_{1\leq i\leq m}$ of (5.3), and the max-plus polynomial matrices $(\phi_i)_{1\leq i\leq n}$ as defined above. Then,*

- *For all shrinking matrices $(Q_i)_{1\leq i\leq m}$, there exists $\delta_0 > 0$ such that $\big((M_i)_{1\leq i\leq m}, (Q_i)_{1\leq i\leq m}, \delta_0\big)$ is a non-empty shrunk solution of (5.3) if, and only if, $(Q_i)_{1\leq i\leq m}$ is a solution of (5.4) in $\mathbb{N}$. Given $(M_i)_{1\leq i\leq m}$ and $(Q_i)_{1\leq i\leq m}$, one can compute the maximum $\delta_0 > 0$ to satisfy this property.*
- *$\big((M_i)_{1\leq i\leq m}, (Q_i)_{1\leq i\leq m}, \delta_0\big)$ is the greatest shrunk solution of (5.3) if, and only if $(Q_i)_{1\leq i\leq m}$ is the least solution of (5.4) in $\mathbb{N}$.*
- *If (5.4) has a solution $(Q_i)_{1\leq i\leq m}$, then for any shrinking matrices $R_{n+1}, \ldots, R_m$ such that $\mathcal{P}(R_j) \geq \mathcal{P}(Q_j)$ for $n+1 \leq j \leq m$, there exist $R_1, \ldots, R_n$ such that $(R_i)_{1\leq i\leq m}$ is the least shrunk solution of (5.4), computable in polynomial time.*

The first two statements follow directly from the previous paragraph and Corollary 3.4.15. The computation of $\delta_0 > 0$ is explained in Section 3.4. The third statement follows easily from Theorem 5.4.1 given in the next section, where we study the efficient computation of the solutions to fixpoint equations in the max-plus algebra, as in Equation (5.4).

5.4 Max-Plus Algebra

5.4.1 Max-plus equations

In PSMs, formal expressions using maximization and sum are manipulated. The set $\mathbb{R}_{\geq 0}$ endowed with these operations is called the *max-plus algebra*. There is a well-established theory on solving equations in this algebra, with applications to discrete-event systems [BCOQ92]. The purpose of this section is to show how to solve polynomial fixpoint equations in the max-plus algebra.

Let $k_1, \ldots, k_n, k_{n+1}, \ldots, k_{n+n'}$ be parameters, and $\phi_1, \ldots, \phi_n$ be max-plus polynomials. We are interested in computing solutions of fixpoint equations of the following form:

$$k_i = \phi_i(k_1, \ldots, k_n, k_{n+1}, \ldots, k_{n+n'}), \quad \forall 1 \leq i \leq n. \tag{5.6}$$

Notice that variables $k_{n+1}, \ldots, k_{n+n'}$ only appear at the right hand side of the equation. Equation (5.6) defines a *non-linear* equation (polynomials ϕ_i have arbitrary degrees). We call these equations *max-plus polynomial fixpoint equations*. The equations between parameters that we derived

in the previous section fall into this category. Although Tarski's Theorem [Tar55] guarantees the existence of fixpoint solutions in $\mathbb{N} \cup \{\infty\}$, we are interested in *finite* solutions, *i.e.*, solutions in $\mathbb{N}$ which is not a complete lattice.

Theorem 5.4.1. *The existence of a solution of a given max-plus polynomial fixpoint equation is decidable in polynomial time in the size of the equation.*

Moreover, if there is a solution $\mathbf{v}$ *in* $\mathbb{N}$ *to a given equation* E*, then for any values* $v'_{n+1}, \ldots, v'_{n+n'} \in \mathbb{N}$ *where* $v_{n+i} > 0 \Rightarrow v'_{n+i} > 0$ *for all* $1 \leq i \leq n'$*, equation* E *with the additional constraints* $\{k_{n+i} = v_{n+i}\}_{1 \leq i \leq n'}$ *has a least solution, computable in polynomial time.*

As in the previous section, we assume that expressions can be shared in equations given as input to the above theorem. Such a data structure is detailed in the next subsection. The second point of the theorem states that the existence of solutions does not depend on the exact values of the unconstrained variables, but only on their positiveness.

These results rely on an analysis of max-plus graphs, that we associate to max-plus equations. The rest of this section defines these graphs and gives an algorithm to solve these equations.

5.4.2 Max-Plus Graphs

Let $\mathbf{k}$ be a set of parameters. A *max-plus graph* G *with parameters* $\mathbf{k}$ is a directed graph (V, E), where V is the set of *nodes*, and $E \subseteq V \times V$ the set of *arcs*. The node set V is partitioned into $V = \mathbf{k} \cup \mathbf{N} \cup \mathbf{Max} \cup \mathbf{Plus}$. There is one node for each parameter, and also some additional nodes labelled by natural numbers, and others labelled either by max or by plus. The arcs satisfy the constraints that each node labelled by max or plus has exactly two incoming arcs and one outgoing arc; each (directed) cycle contains at least one node in $\mathbf{k}$; and the nodes $\mathbf{N}$ do not have incoming arcs. We identify nodes $n \in \mathbf{N}$ with the natural number they represent. We will have at most one node labelled by each natural number.

Intuitively, a max-plus graph encodes the relations between the parameters $\mathbf{k}$, where a directed path from parameter k to k' means that k' is greater than or equal to k in any solution. An *extended valuation* $\overline{v}$ is the extension of a parameter valuation v to all nodes of G.

Definition 5.4.2. *A valuation* $v : \mathbf{k} \to \mathbb{N}$ *is a* solution *of a max-plus graph* G *with parameters* $\mathbf{k}$ *if there exists some extended valuation* $\overline{v}$*, that satisfies the following conditions.*

- *For all* $k \in \mathbf{k}$, $\overline{v}(k) = \max(\overline{v}(k'))$ *where the max is over all predecessors* k' *of* k,
- *For all* $n \in \boldsymbol{N}$, $\overline{v}(n) = n$.
- *For all* $p \in \boldsymbol{Plus}$, $\overline{v}(p) = \overline{v}(p') + \overline{v}(p'')$, *where* p' *and* p'' *are the (unique) predecessors of* p,
- *For all* $m \in \boldsymbol{Max}$, $\overline{v}(m) = \max(\overline{v}(m'), \overline{v}(m''))$ *where* m' *and* m'' *are the (unique) predecessors of* m.

Note that if we are interested in a positive integer solution of a max-plus graph, we can simply add one edge from each parameter k to the node 1. We extend the order $\leq$ to vector of numbers as $(a_i)_{i \in I} \leq (b_i)_{i \in I}$ iff $a_i \leq b_i$ for all $i \in I$.

Given a graph $G = (V, E)$, a *path* from node v_1 to node v_k is a sequence $v_1 v_2 \ldots v_k$ of nodes where $(v_i, v_{i+1}) \in E$ for all $1 \leq i \leq k-1$. A *simple cycle* is a path $v_1 \ldots v_k$ such that $v_1 = v_k$ and nodes $v_1, \ldots, v_{k-1}$ are pairwise distinct. A node v' is *reachable* from a node v if there is a

path starting at v and ending at v'. For two nodes $v, v' \in V$, we write $v \xrightarrow{+} v'$, if there is a path $v = v_1 \dots v_k = v'$ such that $k \geq 2$, and $v_i \in \mathbf{Plus}$ for some $i \in \{1, \dots, k\}$.

The following lemma gives a graph theoretical characterization of max-plus graphs that have solutions. A simple cycle of a max-plus graph is a *bad cycle* if it contains at least one **Plus** node, and at least one node that is reachable from a node $n \in \mathbf{N}$ with $n \geq 1$. A *contradicting path* is a path from k to l, for some $k, l \in \mathbf{N}$ with $k > l$.

Lemma 5.4.3. *A max-plus graph G with parameters $\mathbf{k}$ has a solution if and only if it has no bad cycle or contradicting path. Moreover, if G has a solution, then it has a least solution which can be computed in polynomial time in the size of G.*

Proof. Clearly, if G has contradicting paths, it has no solution. Let us show that if G has a bad cycle, then it does not have any solution.

Consider a simple bad cycle c in G that contains a plus node p, and suppose there is an extended solution $\overline{v}$. We know that p has one predecessor in p' in c, and another one outside, let us call it p''. By hypothesis, all nodes of c are reachable from a node $n \geq 1$, which implies $\overline{v}(p'') \geq 1$. We have $\overline{v}(p) \geq \overline{v}(p') + \overline{v}(p'') \geq \overline{v}(p') + 1$. But since c is a cycle, combining inequalities satisfied by $\overline{v}$ along c, we get $\overline{v}(p') \geq \overline{v}(p)$, therefore $\overline{v}(p') \geq \overline{v}(p') + 1$, which is a contradiction.

We now prove that if no such cycle exists, then G has a solution, which we will construct explicitly. Consider first all nodes W of G that are not reachable from any node $n \in \mathbf{N}$ with $n \geq 1$. We can safely assign the value 0 to all the nodes of W (this doesn't contradict the max-plus graph since none of these nodes are reachable from a positive number). Then, we replace all nodes of W by the node $0 \in \mathbf{Nat}$. Let us call G' the graph obtained from G in this manner. Clearly, if G' has a solution, so does G, by extending the solution with $W \cap \mathbf{k} \mapsto 0$. Moreover, G' can be computed in polynomial time.

If $\mathbf{k} \subseteq W$ then we are done. Otherwise, let us assume w.l.o.g. that all the nodes of G are reachable from a positive number. We are now looking for a solution in $\mathbb{N}_{>0}$. Then, by hypothesis, the relation $\xrightarrow{+}$ is anti-symmetric. We partition the set of nodes of G into $C_0, C_1, \dots, C_m$ such that C_0 is the minimum for the relation $\xrightarrow{+}$, and each C_i is the set of nodes whose all $\xrightarrow{+}$-predecessors are in $C_0 \cup \dots \cup C_{i-1}$ and that has at least one $\xrightarrow{+}$-predecessor in C_{i-1}. This partition can be seen as a topological sort for the relation $\xrightarrow{+}$.

Observe that C_0 is not empty by hypothesis (in fact, there is at least one node that is reachable from a positive integer and there is no bad cycle) and that any plus node of C_i have both its predecessors in $C_0 \cup \dots \cup C_{i-1}$. Also, there is no path in G from C_i to C_j for $j < i$ by construction. Let $G[C_i]$ denote the graph G restricted to nodes of C_i. For all $1 \leq i \leq m$, given a *lower bound* $\iota_i \colon C_i \to \mathbb{N}_{>0}$, there is a unique solution v_i of $G[C_i]$ such that $\iota_i(v) \leq v_i(v)$ for all $v \in C_i$. In fact, the solutions of $G[C_i]$ which respect ι_i are exactly the solutions of the max-plus linear equation $x = Ax \oplus b$, defined by $A_{k,l} = 0$ if there is an arc (l, k) in $G[C_i]$ and $A_{k,l} = -\infty$ otherwise; and vector b is defined as $b_k = \iota_i(k)$ for all k. [3] Solutions to this linear fixpoint equations exist in $\mathbb{N}_{>0}$, and A^*b is one, and is in fact the least solution, where A^* is defined as $A^*_{k,l} = 0$ if there is path from l to k in $G[C_i]$, and $-\infty$ otherwise. This can be computed in polynomial time (see Section 3 of [BCOQ92]).

We will define v iteratively for each C_i, $i \geq 0$. For $i = 0$, we let v_0 be the least solution of $G[C_0]$ for ι_0 defined by $\iota_0(n) = n$ for $n \in \mathbf{N} \cap C_0$ and $\iota_0(v) = 1$ for all $v \in C_0 \setminus \mathbf{N}$. At step $i \geq 0$,

[3] In max-plus algebra, max is the addition and is denoted by $\oplus$, whereas the sum is the multiplication and is denoted by $\otimes$. The product of a matrix and a vector is defined as usual, where multiplication is $\otimes$ and addition is $\oplus$.

suppose we are given valuations $v_0, \ldots, v_{i-1}$ and a lower bound ι_i. We compute v_i as the least solution of $G[C_i]$ that respects ι_i, and we define ι_{i+1} on C_{i+1} as follows. For all $u \in C_{i+1} \cap \mathbf{k}$, we let $\iota_{i+1}(u) = \max_{v:(v,u)\in E \wedge v \in C_0 \cup \ldots \cup C_i} v(v)$; for all $u \in C_{i+1} \cap \mathbf{Max}$, we let $\iota_{i+1}(u) = \max(v(v_1), v(v_2))$ where $v_1, v_2 \in C_0 \cup \ldots \cup C_i$ by construction; and for all $u \in C_{i+1} \cap \mathbf{Plus}$, we let $\iota_{i+1}(u) = v(v_1) + v(v_2)$ where $v_1, v_2 \in C_0 \cup \ldots \cup C_i$. (Note that $\mathbf{N} \subseteq C_0$). Clearly, any solution of G that extends $v|_{C_0 \cup \ldots \cup C_i}$ must satisfy this lower bound.

By construction, v defines a solution. Moreover, we show that this is the least solution. In fact, since this is a finite solution, the least solution v' given by Tarski's theorem is also finite. But we defined v_0 as the least solution in C_0, so that there is no solution of G whose values in C_0 is less than v_0. This means $v'|_{C_0} \geq v_0$. We show by induction that the lower bound ι_i is satisfied by $v'|_{C_i}$, for all i, so that we get $v'|_{C_i} \geq v_i$ by construction of v_i. Hence, $v' = v$. □

We now define max-plus graphs that encode the solutions of equations of the form (5.6). First, for a max-plus polynomial ϕ, let us define graph $G(\phi)$ associated to ϕ, as follows. Graph $G(\phi)$ is the syntactic binary tree of the expression ϕ, where the leaves are either constants or elements of $\mathbf{k}$, and each internal node corresponds to either plus or max, and joins the subtrees of the corresponding subexpressions. The root corresponds to the whole expression ϕ. We direct the edges of the tree bottom up. Now, define the *max-plus graph G associated to equation (5.6)* as the union of graphs $G(\phi_i)$ associated to each ϕ_i, where all nodes corresponding to same parameters are merged together. Moreover, an arc is added from the root node of each $G(\phi_i)$ to node k_i, and from k_i to the root. Notice that any cycle we create in this manner contains a parameter node. Observe that in our construction, each max- or plus-node of G corresponds to a unique subexpression in one of the ϕ_i's. The following lemma states that this graph encodes precisely the solutions of equation (5.6).

Lemma 5.4.4. *Let G be the max-plus graph associated to Equation* (5.6)*. A valuation $v\colon \mathbf{k} \to \mathbb{N}$ is a solution of G if, and only if it is a solution of* (5.6)*. Therefore, v is the least solution of G if, and only if it is the least finite solution of* (5.6)*.*

Proof. As noted above, there is a correspondence between the nodes labelled max or plus with the subexpressions. We use this to show how a solution of G can be seen as a solution of (5.6) and vice versa. Suppose $\overline{v}$ is an extended solution that proves that v is a solution of G. Then, this defines a solution of (5.6) where $v(\mathbf{k})$ is the values given to $\mathbf{k}$, and $v(V \setminus \mathbf{k})$ define the values of the subexpressions in ϕ_i's. In fact, by construction, these are the "intermediate" values calculated in the subexpressions when one evaluates each ϕ_i. Using the same correspondence, one can see that a solution of (5.6) yields a solution of G whose extension correspond to these intermediate values. □

Proof of Theorem 5.4.1. The first statement of Theorem 5.4.1 now follows directly from Lemmas 5.4.3 and 5.4.4. For the second statement, define equation (E) from equation (5.6) by adding equality constraints $k_{n+i} = v_{n+i}$ for all $1 \leq i \leq n'$, for any $v_{n+1}, \ldots, v_{n+n'} \in \mathbb{N}_{>0}$. Assume that (5.6) has a solution where $k_{n+1}, \ldots, k_{n+n'}$ are given positive values. We show that then, the max-plus graph G' associated with equation (E), does not have bad cycles or contradicting paths, which means that it has a solution by above lemmas. In fact, in the max-plus graph G of (5.6), no cycle that is reachable from a node k_{n+i} contains a plus node, since this would contradict the existence of a solution with a positive value for k_{n+i}. Therefore, when we add the additional constraint $k_{n+i} = v_{n+i}$ we do not create any new bad cycles or contradicting paths. The result follows. □

5.5 Deciding shrinkability

We now apply the results we developed in previous sections to shrinkability.

5.5.1 Simulation-Shrinkability

We fix a closed timed automaton $\mathcal{A} = (\mathcal{L}, l_0, \mathcal{C}, \Sigma, E)$ with distinct labels, and a finite automaton $\mathcal{F}$ on the same alphabet Σ such that $[\![\mathcal{F}]\!] \sqsubseteq_{\text{t.a.}} [\![\mathcal{A}]\!]$. For any edge with label $\sigma \in \Sigma$ and guard g_σ, let G_σ be the DBM that represents $[\![g_\sigma]\!]$, and R_σ be the reset set.

Computation of the Simulator Sets Since all edge labels are distinct, the simulator set of each state f of $\mathcal{F}$ in $[\![\mathcal{A}]\!]$ can be expressed as the greatest fixpoint of the following equation:

$$S_f = \bigcap_{\sigma\in\Sigma} \bigcap_{f \xrightarrow{\sigma} f'} \mathsf{Pre}_{\mathsf{time}}\left(\mathsf{Unreset}_{R_\sigma}(S_{f'}) \cap [\![G_\sigma]\!]\right), \tag{5.7}$$

for all states f of $\mathcal{F}$, where $(S_f)_{f\in\mathcal{F}}$ are unknown sets of states of $\mathcal{A}$. The greatest fixpoint is well-defined since the operator on the right, denoted Ω, is non-decreasing.

We will use properties of the region equivalence in a timed automaton without formalizing this well-known construction (for that, we refer to [AD94]). Regions refine the largest time-abstract bisimulation in a timed automaton; therefore, if $(S_f)_f$ are finite unions of regions, then $(S'_f)_f = \Omega((S_f)_f)$ are also finite unions of regions. In particular, the largest fixpoint of (5.7) can be computed by a finite number of iterations of operator Ω, after having initialized all sets with the full set of states of $\mathcal{A}$. We also notice that if $(S_f)_f$ are convex sets, then so are $(S'_f)_f = \Omega((S_f)_f)$.

For every i, we write $(S^i_f)_f$ for the sets of states obtained after i iterations of Ω in the above-mentioned iterative computation. From the above discussion, the two following properties hold:

- for every state f of $\mathcal{F}$, ta-sim$_{[\![\mathcal{A}]\!]}(f) = \lim_{i\to\infty} S^i_f = S^{i_0}_f$ for some i_0;
- for every i, for every f, S^i_f is a finite convex union of regions.

We therefore deduce that the simulator set of each state f of $\mathcal{F}$ in $[\![\mathcal{A}]\!]$ can be expressed as the greatest fixpoint of the following equation on DBMs:

$$[\![M_f]\!] = \bigcap_{\sigma\in\Sigma} \bigcap_{f \xrightarrow{\sigma} f'} \mathsf{Pre}_{\mathsf{time}}\left(\mathsf{Unreset}_{R_\sigma}([\![M_{f'}]\!]) \cap [\![G_\sigma]\!]\right), \tag{5.8}$$

for all states f of $\mathcal{F}$, where $(M_f)_f$ are unknown DBMs.

We now argue that the simulator sets can be computed in time pseudo-polynomial in $\mathcal{A}$ and $\mathcal{F}$. As argued above, one can compute this greatest fixpoint by initializing all M_f to unconstrained DBMs, and then iteratively computing approximations applying (5.8) until a fixpoint is reached. If M^i_f is the i-th iterative DBM for state f, we obviously have that $[\![M^i_f]\!] = S^i_f$. As each computed DBM represents a finite union of regions, it can be defined with constraints using integer constants between $-C$ and C, and all the computed normalized DBMs only use integer constants in $[-C\cdot|\mathcal{C}_0|, C\cdot|\mathcal{C}_0|] \cup \{-\infty, \infty\}$.

So, at each iteration, either the M_f's do not change, in which case the fixpoint has been reached, or some constant is decreased by at least one. Thus, fixpoint must be reached in at most $O((C\cdot|\mathcal{C}_0|)\cdot|\mathcal{F}|\cdot|\mathcal{C}_0|^2)$ iterations, and each computation takes time $O(|\mathcal{F}|\cdot|\mathcal{C}_0|^3)$, since the expression inside the intersection in (5.8) is computed, and all DBMs need be normalized for each edge of $\mathcal{F}$.

Globally, the simulator sets for all f's can therefore be computed in time $O(C \cdot |\mathcal{F}|^2 \cdot |\mathcal{C}_0|^6)$, which is then pseudo-polynomial.

Computing Shrunk Simulator Sets When They Exist Consider the greatest solution $(M_f)_f$ of (5.8). Including the G_σ's in the unknown DBMs, Equation (5.8) can be seen as an instantiation of Equation (5.3) (page 76) over DBMs $(M_f)_f \cup (G_\sigma)_\sigma$.

Solving simulation-shrinkability w.r.t. $\mathcal{F}$ consists in deciding if for some shrinking of the guards G_σ, there exist simulator sets that are shrinkings of the sets M_f's. So, solving simulation-shrinkability w.r.t. $\mathcal{F}$ means deciding whether (5.8) has a shrunk solution with respect to $(M_f)_f \cup (G_\sigma)_\sigma$ where the shrinking matrices of G_σ's are positive.[4] This can be decided by Lemma 5.3.3.

Simulation-shrinkability does not depend on how much the guards are shrunk. In fact, since G_σ's are unconstrained in (5.7), if there is a shrunk solution to (5.7) with positive shrinking matrices for G_σ's, then for any shrinking matrices $(K_\sigma)_\sigma$, Lemma 5.3.3 provides a (greatest) shrunk solution where the shrinking matrices for $(G_\sigma)_\sigma$ are fixed to $(K_\sigma)_\sigma$. Therefore, either all positive integer vectors $\mathbf{k}$, which yield normalized guards, witness the shrinkability of $\mathcal{A}$ (into $\mathcal{A}_{-\mathbf{k}\delta}$), or $\mathcal{A}$ is not simulation-shrinkable w.r.t. $\mathcal{F}$ for any value of $\mathbf{k}$.

Furthermore, one can also require *initial* simulation between $\mathcal{F}$ and a shrinking of $\mathcal{A}$, *i.e.* the initial state of $\mathcal{A}_{-\mathbf{k}\delta}$ should simulate the initial state f_0 of $\mathcal{F}$. In this case, it suffices to compute the shrunk simulator sets as above (if these exist), and then check whether the shrinking of the set M_{f_0} contains the valuation $\mathbf{0}$. This can be done efficiently, since it suffices to intersect the shrinking with this valuation and check for emptiness.

Now, solving simulation-shrinkability w.r.t. $\mathcal{F}$ only takes time polynomial in the size of the equation. So the overall complexity is pseudo-polynomial in $\mathcal{A}$ and $\mathcal{F}$. Note that, one can check simulation-shrinkability w.r.t. $\mathcal{A}$, which requires the shrinking to time-abstract simulate $\mathcal{A}$, by chosing $\mathcal{F}$ as a time-abstract bisimulation quotient of $\mathcal{A}$. In this case, $\mathcal{F}$ has exponential size, so the procedure takes exponential time. Nevertheless, minimization can be used to compute the coarsest bisimulation quotient as in [TY01], which yields small equations in practice (See Chapter 6).

5.5.2 Non-blocking-Shrinkability

We fix a closed non-blocking timed automaton $\mathcal{A} = (\mathcal{L}, l_0, \mathcal{C}, \Sigma, E)$. For any edge with label $\sigma \in \Sigma$ and guard g_σ, let G_σ be the DBM that represents $[\![g_\sigma]\!]$, and R_σ be the reset set. The following equation characterizes non-blockingness:

$$\forall \sigma \in \Sigma, \quad [\![G_\sigma]\!] \subseteq \bigcup_{\sigma' : (\sigma,\sigma') \in \Sigma_{E \circ E}} \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!])), \tag{5.9}$$

where we let $\Sigma_{E \circ E} = \{(\sigma, \sigma') \mid \exists l, l', l'' \in \mathcal{L},\ l \xrightarrow{g_\sigma, \sigma, R_\sigma} l' \xrightarrow{g_{\sigma'}, \sigma', R_{\sigma'}} l'' \in E\}$, that is the set of pairs of labels of consecutive transitions in $\mathcal{A}$. We rewrite this equivalently as follows.

$$\forall \sigma \in \Sigma, \quad [\![G_\sigma]\!] = \bigcup_{\sigma' : (\sigma,\sigma') \in \Sigma_{E \circ E}} \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!])) \cap [\![G_\sigma]\!], \tag{5.10}$$

Now, $\mathcal{A}$ is shrinkable w.r.t. non-blockingness if, and only if, this equation has a shrunk solution w.r.t. $(G_\sigma)_\sigma$. We can unfortunately not directly use our general results on shrunk solutions since our

[4] Let us call a shrinking matrix *positive*, if all its off-diagonal components are positive.

equation contains a union. We instead apply transformations to this equation in order to remove the union. We start by rewriting the above equation as follows:

$$\begin{aligned} \forall \sigma \in \Sigma, \quad [\![G_\sigma]\!] &= \bigcup_{\sigma' : (\sigma,\sigma') \in \Sigma_{E \circ E}} [\![M_{\sigma,\sigma'}]\!] \\ \forall \sigma, \sigma' \in \Sigma, \quad [\![M_{\sigma,\sigma'}]\!] &= \mathsf{Unreset}_{R_\sigma}(\mathsf{Pre}_{\mathsf{time}}([\![G_{\sigma'}]\!])) \cap [\![G_\sigma]\!] \end{aligned} \tag{5.11}$$

Fix a solution $(G_\sigma)_\sigma \cup (M_{\sigma,\sigma'})_{\sigma,\sigma'}$, which exists again by the non-blockingness assumption. We will solve the max-plus equation corresponding to the second part of (5.11) by Lemma 5.3.3, but we first add to this equation some inequalities which "encode" the first part of (5.11). We use the following technical lemma to choose these inequalities.

Lemma 5.5.1. *Let $C_1, \ldots, C_b$ and D be normalized DBMs satisfying $[\![D]\!] = \bigcup_{1 \le i \le b} [\![C_i]\!]$ and $P_1, \ldots, P_b$ and Q shrinking matrices s.t. for some $\delta_0 > 0$, $D - \delta Q$ and $C_i - \delta P_i$ are normalized for all $\delta \in [0, \delta_0]$. Then, one can decide the existence of (and compute the maximum) $\delta_1 > 0$ s.t. $[\![D - \delta Q]\!] = \bigcup_{1 \le i \le b} [\![C_i - \delta P_i]\!]$ for all $0 < \delta \le \min(\delta_0, \delta_1)$, in polynomial space and in time $O(|\mathcal{C}_0|^{2b} p(|\mathcal{A}|))$, where $p(\cdot)$ is a polynomial.*

Moreover, in this case, for all shrinking matrices $Q', P'_1, \ldots, P'_b$ s.t. $Q_{x,y} \bowtie (P_i)_{x,y} \Leftrightarrow Q'_{x,y} \bowtie (P'_i)_{x,y}$ and $(P_i)_{x,y} \bowtie (P_j)_{x,y} \Leftrightarrow (P'_i)_{x,y} \bowtie (P'_j)_{x,y}$ for all $i, j \in \{1, \ldots, b\}$, $x, y \in \mathcal{C}_0$ and $\bowtie \in \{<, =\}$, it holds

$$[\![D - \delta Q']\!] = \bigcup_{1 \le i \le b} [\![C_i - \delta P'_i]\!],$$

for all small enough $\delta > 0$.

Proof. One can verify in polynomial time, whether for all $i \in \{1, \ldots, b\}$, $[\![C_i - \delta P_i]\!] \subseteq [\![D - \delta Q]\!]$ for all small enough $\delta \ge 0$: It suffices to check, for all x, y, whether $(C_i - \delta P_i)_{x,y} \le (D - \delta Q)_{x,y}$ for small enough $\delta \ge 0$, which holds if either $(C_i)_{x,y} < D_{x,y}$ or $(C_i)_{x,y} = D_{x,y}$ and $(P_i)_{x,y} \ge Q_{x,y}$. In the former case, we need to choose δ_1 so that $\delta(Q_{x,y} - (P_i)_{x,y}) \le D_{x,y} - (C_i)_{x,y}$ for all $0 \le \delta \le \delta_1$, whereas the latter always holds.

It remains to verify that $[\![D - \delta Q]\!] \subseteq \bigcup_{1 \le i \le n} [\![C_i - \delta P_i]\!]$. This holds if and only if

$$\Big(\bigcup_{1 \le i \le b} [\![C_i - \delta P_i]\!] \Big)^c \cap [\![D - \delta Q]\!] = \bigcap_{1 \le i \le b} \bigcup_{(x,y) \in (\mathcal{C} \cup \{0\})^2} [\![x - y > (C_i - \delta P_i)_{x,y}]\!] \cap [\![D - \delta Q]\!] = \emptyset.$$

which is true if and only if for all $(x^{(1)}, y^{(1)}), \ldots, (x^{(b)}, y^{(b)}) \in (\mathcal{C} \cup \{0\})^2$, $\bigcap_{1 \le i \le b} ([\![x^{(i)} - y^{(i)} > (C_i - \delta P_i)_{x^{(i)}, y^{(i)}}]\!] \cap [\![D - \delta Q]\!]) = \emptyset$. But there are less than $(|\mathcal{C}_0|)^{2b}$ such terms, which are conjunctions of $b + 1$ DBMs, so the emptiness of each term can be checked in polynomial time, as described above. The overall time complexity is then $O((|\mathcal{C}_0|)^{2b} p(|\mathcal{A}|))$. But this verification can be carried out in polynomial space since each term can be checked independently. Note that each term gives an upper bound on δ_1, so the equality holds choosing δ_1 as the minimum of these.

The last statement follows from the fact that the emptiness, for all small enough $\delta \ge 0$, of the disjuncts above only depends on the order between the parameters. □

Note that checking equality between a zone and a union of zones is a difficult problem in general; some heuristics were suggested in [DHLP06].

The second point of the lemma says that the satisfaction of the first part of (5.11) by a shrunk solution only depends on the relative ordering of the components of the shrinking matrices. Therefore, we only need to guess the ordering between all parameters (there is at least one if there exists a shrunk solution), and solve the second part of (5.11) augmented with these guessed (in)equalities.

Formally, let Φ be the max-plus equation corresponding to the second part of (5.11), as defined in Section 5.3. Let $\mathbf{k}'$ denote the set of all parameters that appear in Φ (there is one parameter per element of each matrix G_σ and $M_{\sigma,\sigma'}$). Notice that $\mathbf{k}'$ has size $O\big((|\mathcal{C}_0| \cdot |\mathcal{L}| \cdot b)^2\big)$, where b is the maximal number of outgoing edges in $\mathcal{A}$, and that Φ has size polynomial in the size of $\mathcal{A}$. Φ is a conjunction of equations $k = \phi_k(\mathbf{k}')$ for all $k \in \mathbf{k}'$. For all pairs $k, l \in \mathbf{k}'$, we guess a relation among $\{<, =, >\}$, and define equation Φ' by adding these relations to Φ. This can be done, for the case $k = l$, by replacing the constraints on k and l respectively by $k = \max(\phi_k(\mathbf{k}'), l)$ and $l = \max(\phi_k(\mathbf{k}'), k)$, and in the case $k > l$, by replacing the constraint on k by $k = \max(\phi_k(\mathbf{k}'), l+1)$. Notice that Φ' is obtained from Φ in polynomial time and with a polynomial number of guesses. We then solve Φ' using Theorem 5.4.1. If we find a solution, say $(P_\sigma)_\sigma \cup (P_{\sigma,\sigma'})_{\sigma,\sigma'}$, we verify that $[\![G_\sigma - \delta P_\sigma]\!] = \bigcup_{\sigma'} [\![M_{\sigma,\sigma'} - \delta P_{\sigma,\sigma'}]\!]$ for small δ, for all pairs $(\sigma, \sigma') \in \Sigma_{E \circ E}$, in time $O(|\mathcal{C}_0|^{2b} p(|\mathcal{A}|))$ and in polynomial space by Lemma 5.5.1. We accept if all verifications succeed and reject otherwise. If accepted, any solution provides a shrunk solution of (5.11), by Lemma 5.3.3. Conversely, if there is a shrunk solution of (5.11), then, Φ' can be constructed for the guesses corresponding to this solution, and by Lemma 5.5.1, Φ' has a solution. If b is fixed, this procedure is in NP. Otherwise, instead of making guesses, we can deterministically try all possible guesses (the number of possible guesses is $O(2^{(|\mathcal{C}| \cdot |\mathcal{L}| \cdot b)^2})$ and verify in polynomial space, so the procedure is then in PSPACE.

Finally, to decide strong shrinkability, one can first compute the least parameters $\mathbf{k}$ and δ_0 for non-blocking-shrinking, then check simulation-shrinkability since the latter does not depend on exact values of $\mathbf{k}$ and δ_0.

5.6 Conclusion

In this chapter, we proposed a new framework for robustness analysis against guard shrinkings, that can detect timed automata whose behaviours depend on the ability of the system to act at the boundaries of the guards. Such timed automata are vulnerable to loss of time-abstract behaviour when implemented. Theoretically, shrinkability analysis is formulated as a parameter synthesis problem. We showed how the fixpoint equations between DBMs that encode the desired properties are reduced to fixpoint equations in the max-plus algebra between the parameters. Despite the relative complexity of the proofs and algorithms, the simulation-shrinkability works well in practice; the next chapter presents a tool and experimental results.

An important future work is the study of arbitrary fixpoint equations between shrunk DBMs, such as those including arbitrary use of unions. If we are able to treat such equations, then simulation-shrinkability without the distinct edge labels hypothesis could be solved. This would allow one to write more abstract specifications, and ease the complexity. In fact, as our experimental results indicate, the bottleneck is the size of the finite automaton (denoted $\mathcal{F}$) with respect to which simulation-shrinkability is checked. One could also check shrinkability w.r.t. a (timed) temporal logic formula.

We believe the shrunk DBMs we introduced can be useful in solving other problems in timed automata, involving small imprecisions. We indeed re-use this data structure to solve the robust controller synthesis problem in two differents settings; these are studied in Part III of this thesis.

Chapter 6

The Shrinktech tool

6.1 Introduction

Chapter 5 presented algorithms and theoretical results on shrinkability analysis on timed automata. This chapter presents the tool `shrinktech`, in which the simulation-shrinkability algorithm is implemented, and reports experimental results.

The contents of this chapter will appear in [San13].

6.2 Shrinktech

Given a timed automaton, the tool `shrinktech` either finds a counter-example to simulation-shrinkability, such as a path or a cycle that cannot be executed by *any* shrinking of the automaton, whatever the value of δ's are, or outputs a shrinking of the timed automaton that witnesses the shrinkability.

The input to the tool is Kronos timed automata format[1]. The format allows one to either specify a timed automaton directly (as in Definition 2.2.1), or to describe a *network* of timed automata, that is, several timed automata synchronizing on common actions. Figure 6.2 shows an overview of the tool. To check the shrinkability of a timed automaton, the user can either provide a finite automaton $\mathcal{F}$, or let `shrinktech` compute the full finite bisimilarity graph using Kronos[2]. Note that if the full bisimilarity graph is too big, one can also try to shrink with respect to a portion of it, or with respect to a randomly generated trace. This is to be compared with bounded model-checking, which is useful for detecting bugs, but also for "partially" proving the correctness of a system. The tool comes with scripts to compute the bisimilarity graph, extract some (random) portion of it, and generate random executions.

The tool `shrinktech` can be used for several kinds of systems modelled by timed automata. We believe it can be used mainly for two purposes:

1. Robustness analysis, to find out whether the behaviour of the system is preserved when the time bounds are disturbed (shrunk).
2. Deriving implementations from timed automata. In fact, the behaviour of a shrunk timed automaton is included in that of the initial model in presence of imprecisions. So lower and upper bounds on the delays can be "shrunk" in the implementation to guarantee that these will be respected despite imprecisions. This approach will be detailed in Chapter 11.

Implementation details and availability. The tool is implemented in C++ and the source code has about 5Klocs. It uses the Uppaal DBM library [3], and extends it to shrunk DBMs presented in

[1] Kronos is a model-checker for timed automata [BDM+98].
[2] Kronos can *minimize* the region graph of a timed automaton, as described in [TY01].
[3] `http://people.cs.aau.dk/~adavid/UDBM/`

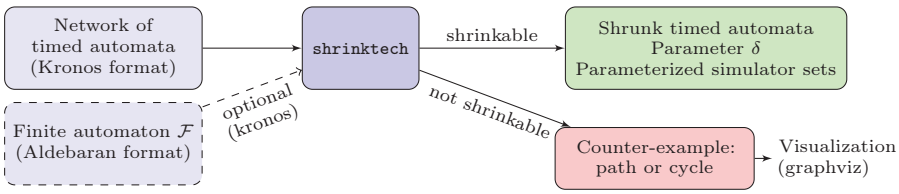


Figure 6.1: Overview of `shrinktech`.

Chapter 5. The input formats are (networks of) timed automata in the Kronos format, and finite automata in the Aldebaran format[4]. The tool Kronos can be plugged in the tool-chain in order to compute the finite time-abstract bisimilarity graph of a given timed automaton, to be used as the finite automaton $\mathcal{F}$. Shrinktech is open source software and is distributed under GNU General Public Licence 3.0. It is freely available at:

`http://www.lsv.ens-cachan.fr/Software/shrinktech/`

6.3 Experimental Results

We used `shrinktech` on several case studies found in the literature. The table 6.1 summarizes the results. The Lip Synchronization Protocol has been the subject of robustness analysis (by guard enlargement) before [KLP09]. This is an algorithm that synchronizes video and sound streams that arrive in different frequencies. The model is not shrinkable neither for video frames arriving in exact frequency, nor for those arriving within a bounded interval. Observe that the model is shrinkable w.r.t. a small subgraph with 501 nodes, but it is not shrinkable w.r.t. the whole graph which has 4484 nodes. Shrinkable models include Philips Audio Retransmission protocol [DY95], and some asynchronous circuit models. We were able to analyze Fischer's Mutual Exclusion Protocol upto 4 agents; while for 5 agents we could only partially analyze w.r.t. a randomly generated trace. The non-shrinkability of most models is due to equality constraints. In fact, although we only shrink non-punctual guards, some behaviours may still disappear immediately, however small the shrinking parameter is.

Note that some of these models were designed at a level of abstraction where imprecisions were not taken into account. So, our results do not necessarily imply that these systems are not robust, but rather that the present models are not good for direct implementation. This is best illustrated in the Latch Circuit models, where the exact model that extensively uses equalities is not shrinkable, but its relaxation to intervals is. Notice also how most of the circuit models, which define bounds on stabilization times are shrinkable.

6.4 Related Work

Existing verification tools for timed automata may be used for *non-parameterized* robustness checking by modeling explicitly the imprecisions, although this increases the size of the models [AT05]. The semi-algorithm of HyTech was used to synthesize guard enlargement parameters in timed automata

[4]This is a graph description format of the CADP tool suite, also used by Kronos.
See `http://www.inrialpes.fr/vasy/cadp/`

Table 6.1: The column sim-graph is the number of states and the number of transitions of the finite automaton $\mathcal{F}$ w.r.t. which the shrinkability is checked. $\mathcal{L}$, E, and $\mathcal{C}$ denote, respectively, the number of locations, edges, and clocks. The last column is the shrinkability of the given model, as reported by the tool. An asterisk indicates bounded shrinkability, where only a subgraph of the time-abstract bisimulation graph (given by a BFS) or a random trace was used. The tests were performed on an Intel Xeon 2.67 GHz. All models are available on the tool's website.

Model	$\lvert\mathcal{L}\rvert$	$\lvert E\rvert$	$\lvert\mathcal{C}\rvert$	**sim-graph**	**time**	**shr**
Lip-Sync Prot. (Exact)	230	680	5	4000/8350* (subgraph)	9s	No
Lip-Sync Prot. (Interval)	230	680	5	501/1282* (subgraph)	9s	Yes*
Lip-Sync Prot. (Interval)	230	680	5	4484/48049	28s	No
Philips Audio Prot.	446	2097	2	437/2734	46s	Yes
Root Contention Prot.	65	138	6	500/3455* (subgraph)	7s	No
Train Gate Controller	68	199	11	952/8540	34s	No
Fischer's Protocol 3	152	464	3	472/4321	20s	Yes
Fischer's Protocol 4	752	2864	4	4382/65821	310min	Yes
Fischer's Protocol 5	3552	16192	5	10000/10000* (trace)	42s	Yes*
And-Or Circuit	12	20	4	80/497	1.3s	Yes
Flip-Flop Circuit	22	34	5	30/64	0.9s	Yes
Latch Circuit (Interval)	32	77	7	105/364	1.6s	Yes
Latch Circuit (Exact)	32	77	7	100/331	0.6s	No

in [DDR05a]. An extension of Uppaal for robustness against guard enlargement was used in [KLP09]; this feature is no longer available in Uppaal. Note that shrinkability cannot be solved by existing model-checkers for timed automata since we are interested in parameter synthesis so as to ensure time-abstract simulation. Other similar work includes (the undecidable problem of) parameter synthesis in timed automata, where guards are written using parameters, and one tries to find the valuations for which the system satisfies some specification, *e.g.* [AFKS12]. This is difficult to realize due to the large number of parameters (upto millions) and the time-abstract simulation condition we consider. Robustness against large decreases in task execution times using simulation was considered in [ACS10].

6.5 Example: Non-shrinkability

Non-shrinkability of timed automata can be due either to the disappearance of a finite behaviour, or that of an infinite behaviour. We illustrate the first kind of non-shrinkability here; the next section gives an example of the second one.

We consider the timed automaton $\mathcal{A}^3$ of Fig. 6.2. This timed automaton describes a system, which, upon arrival to location A, activates an external device, and then sends a message to the device (at location S). At location A, the device can be activated in two modes: a *simple mode* renders the device ready for execution immediately, but it requires a (slow) buffered communication. An *advanced mode* requires at least one time unit of activation, but has the advantage of supporting direct (and fast) communication. We assume that the device is activated by maintaining a signal long enough on some port: If the signal is maintained long enough (at least 1 time unit), the device switches to advanced mode; otherwise it falls back to the simple mode. Let us further assume that the buffer is active by default, but it is automatically deactivated when (strictly) more than 1 time unit has been spent at A.

The system sends either a direct or a buffered message at location S following the above constraints. Let us assume that the system needs to send the direct message while $x \leq 3$, and that there is no

such constraint for the buffered message.

One may not want to maintain signal that is too long at A; this may be costly, or may consume computation time. On the other hand, if one wants to make sure that the device is activated in the advanced mode, then the signal should be maintained for at least $1 + \delta$, for a suitable δ. Hence, shrinkability analysis makes sense here: *e.g.* the shrunk guard $x - y \geq 1 + \delta$ means exactly that $1 + \delta$ time units have been spent at A.

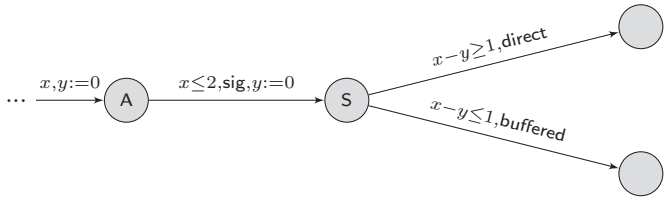


Figure 6.2: Timed automaton $\mathcal{A}^3$.

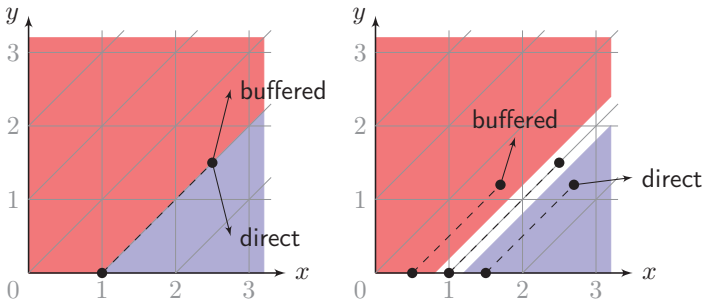


Figure 6.3: The state space of the timed automaton $\mathcal{A}^3$ (on the left), and its shrinking (on the right). The red area is the guard of the edge buffered, while the blue area is that of direct. The initial timed automaton has a branching: from point $x = 1, y = 0$ at location S, the automaton can realize either of the actions. However, this branching is no more possible in the shrunk timed automaton.

Figure 6.3 shows the state space of this timed automaton with and without shrinking. In the original timed automaton, one can see that there is a *branching*: for appropriately chosen delays, after the sig action, both direct and buffered actions are available. In fact, upon arrival to S with $x = 1$, the device is in the advanced mode, and the buffer has not been deactivated. On the other hand, the slightest shrinking of the guards disables such a branching. In any shrunk timed automaton, neither of the actions are enabled if the action sig is taken at $x = 1$. In fact, taking into consideration imprecise measure of time, one cannot be sure, in this case, that the device is in advance mode or the buffer is still active. The safest option would be to realize sig while $x \leq 1 - \delta$ (the buffer is active for sure), or when $x \geq 1 + \delta$ (the device is in advanced mode for sure). Consequently, there is no possible behavior that realizes sig first, and then "decides" whether to choose direct or buffered.

On the other hand, the behaviors sig · direct and sig · buffered are still available in the shrunk timed automaton. So, $\mathcal{A}^3$ is shrinkable w.r.t. these separate paths.

Non-shrinkability is not always due to a finite path that is disabled because of shrinkings. In the

next section, we give an example of a timed automaton whose shrinkings cannot simulate a cycle of the bisimilarity graph, but can simulate any finite unfolding of it.

6.6 Using shrinktech

Simple usage We explain here how to use the `shrinktech` tool on a simple example. We would like to check whether the timed automaton $\mathcal{A}^5$ of Fig. 6.4 is shrinkable.

This automaton can be described in Kronos timed automaton format as given in Fig. 6.5. Assuming Kronos is installed on the system, the simplest way to check shrinkability is the following:

```
> shrinktech automaton.tg
[...]
Starting shrinkability analysis on files automaton.tg and automaton.aut
Warning: Some edges have equality constraints and will not be shrunk (see log file)

Timed automaton is NOT SHRINKABLE.  Counter-example generated in automaton.log and automaton.png
```

Shrinktech first calls kronos (the output is omitted) to compute the time-abstract bisimulation graph of the timed automaton into `automaton.aut`. It then answers that the automaton is not shrinkable. Note also that a warning is generated since not all guards were shrunk (`shrinktech` shrinks all the guards but equality constraints). The counter-example, that is a subgraph of `automaton.aut` that cannot be simulated by any shrinking of `automaton.tg` is generated and it is shown in Fig. 6.5 (this is the file `automaton.png`). The analysis reveals that the cycle BC cannot be simulated in the shrunk automaton, whatever the values of the shrinkings are. In fact, for any shrinking, the cycle can be taken only a finite number of times after which the automaton is blocked.

When the program terminates, the file `automaton.log` contains the text representation of the counter-example, also other useful information such as the shrunk guards and the parameterized simulation sets. The counter-example is also generated as a dot file in `automaton.dot`, and a PNG image is generated in `automaton.png` provided that Graphviz is installed on the system.

Note that `shrinktech` works (when necessary) on a copy of the given timed automaton (the default value is `ta.tg`) because it adds annotations to the edge labels. One can change this file name by passing `-out myfile.tg` as argument to the program.

With a custom finite automaton One can also give a custom finite automaton `automaton1.aut` as an input to the program. For instance, one can try to unfold the loop BC several times to check whether some shrinking of $\mathcal{A}^5$ can at least simulate a finite number of iterations. Consider the finite automaton of Fig. 6.6. This automaton can be simulated by `automaton.aut`, thus also by $\mathcal{A}^5$ (note that the timed automaton must be able to simulate the given finite automaton; an error will be generated otherwise). See also the `-gentrace` option in Section 6.6.1 to randomly generate such a trace.

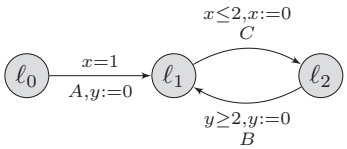


Figure 6.4: Timed automaton $\mathcal{A}^5$.

```
#states 3
#trans  2
#clocks  2
X
Y

state: 0
invar: TRUE
trans:
X = 1 => A; RESET{Y}; goto 1

state: 1
invar: TRUE
trans:
X <= 2 => B; RESET{X}; goto 2

state: 2
invar: TRUE
trans:
Y >= 2 => C; RESET{Y}; goto 1
```

```
des(0,10,6)
(0, "A",1)
(0, "A",2)
(1, "B",3)
(1, "B",4)
(1, "B",5)
(3, "C",1)
(3, "C",2)
(4, "C",2)
(5, "C",1)
(5, "C",2)
```

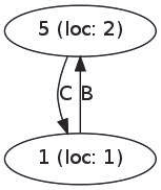


Figure 6.5: The Kronos timed automaton file `automaton.tg` (on the left), the time-abstract bisimulation graph `automaton.aut` generated with Kronos, and the counter-example generated by `shrinktech` (on the right). Nodes are labelled by i(Loc: j) where i is the node of the given automaton $\mathcal{F}$ and j is the unique location of the timed automaton that corresponds to this node i. Edges are labelled by the edge labels of the timed automaton.

```
des(0,7,8)
(0, "A",1)
(1, "B",2)
(2, "C",3)
(3, "B",4)
(4, "C",5)
(5, "B",6)
(6, "C",7)
```

Figure 6.6: Finite automaton `automaton1.aut`

One can run `shrinktech` providing both the timed automaton and the finite automaton:

```
> shrinktech automaton.tg automaton1.aut
Starting shrinkability analysis on files automaton.tg and automaton1.aut
Warning: Some edges have equality constraints and will not be shrunk (see log file)

Timed automaton is SHRINKABLE.

Run again with --simulator_sets option to compute solution.
See automaton.log for details.
```

This time the automaton is shrinkable with respect to the given finite automaton. By default, `shrinktech` does not compute the simulator sets of the shrunk automaton, *i.e.* the set of states of the shrunk timed automaton that can simulate each state of the finite automaton. But this can be done using the `-simulator_sets` option. In this case, the log file will contain the description of the simulator sets. The log file will also report an upper bound on δ below which the simulation holds. Furthermore, a shrinking of the given timed automaton is generated in a local folder named `shrunk`.

Here is an excerpt from the log file.

```
------------------ PARAMETERIZED SHRUNK SIMULATOR SETS ----------------------
-- Substituting delta = 0 gives the simulator sets of the original automaton
-------------------------------------------------------------------------------
Simulator set of the finite automaton node(0)
0 +(p[0])delta <= X <= 1 -(p[2])delta  and
0 +(p[1])delta <= Y  and
X - Y  <= 1 -(p[3])delta

Simulator set of the finite automaton node(1)
0 +(p[4])delta <= X <= 2 -(p[6])delta  and
0 +(p[5])delta <= Y  and
X - Y  <= 2 -(p[7])delta

[...]
```

These describe the set of (parameterized) states that can simulate the nodes 0 and 1 of the finite automaton. Here, $p[i]$'s are positive integers to be determined by shrinkability analysis. The log file also contains an instantiation of these simulator sets for the parameter values computed by `shrinktech` and for the largest possible δ:

```
------------------------- DISCRETE SOLUTION --------------------------
-- Scaling factor: 100
----------------------------------------------------------------------
Discrete simulator set of the finite automaton node(0)
X <= 100 and
X - Y  <= 99

Discrete simulator set of the finite automaton node(1)
X <= 199 and
X - Y  <= 194

[...]
```

Note that we scale here the constants so that we only have to deal with integers, instead of floating point numbers.

See also Section 6.6.1 for additional tools and scripts that can be helpful to extract a finite automaton.

Using multiple files In Kronos timed automaton format, several components communicating by synchronization can be given to describe one large system. In this case, one can simply give `shrinktech` the list of all the components:

```
> shrinktech component1.tg component2.tg .... componentK.tg
```

In this case, `shrinktech` will invoke Kronos to compute the product, and then to compute the bisimulation graph. By default, the product automaton will be written in `ta.tg` and shrinkability analysis will be run on this file. One can modify this target file using the option `-out target.tg`.

6.6.1 Command-line Options and Additional Tools

`-aut` When run with this option, shrinktech will only compute the product of the given timed automata (if more than one is given), and output the time-abstract bisimilarity graph. The output file can be specified using `-out target.aut`; the default is `ta.aut`. This feature uses Kronos.

`-simulator_sets` This option tells `shrinktech` to compute all simulator sets of the shrunk timed automaton, if the given timed automaton is shrinkable. The shrunk timed automata will also be

written in a local directory 'shrunk' in case of shrinkability. This option is disabled by default since it is not always needed.

`-shrink` This option is used to syntactically shrink a given timed automaton. When called `shrinktech -shrink automaton.tg 100`, the program will multiply all constants by 100 and shrink the guards by 1. This is equivalent to shrinking by 1/100; but the time scale is changed so that all constants are integers. By default, the file is output as `automaton-shrunk.tg`, but another target file can be specified with `-out automaton2.tg`.

`-gentrace` Generates a trace, in form of a finite automaton, by a random simulation of the given timed automaton. When called with `shrinktech -gentrace automaton.tg 1000`, this option will output in standard output a random trace of length 1000. One can then either direct this to a file, or specify an output file by `-out automaton.aut`.

`graph_extractor` This is a Python script that can extract a subgraph of a given graph, using DFS or BFS, either deterministically or randomly. The options `-dfs`, `-randdfs` and `-bfs` can be used for this purpose. Run the program without arguments for usage.

Note that CADP can also be useful for instance to check bisimulation, minimize or visualize finite automata (`http://www.inrialpes.fr/vasy/cadp/`).

6.7 Conclusion

We presented `shrinktech`, a software tool compatible with the Kronos model-checker. The tool allows to check the simulation-shrinkability with respect to a given finite automaton. The experimental results show that the tool is capable of treating several case studies found from the literature. The bottleneck is the size of the finite automaton $\mathcal{F}$: the full bisimilarity graph is often costly to compute, and too large graphs require long computation times for the shrinkability analysis. We were nevertheless capable of treating the shrinkability of timed automata with thousands of edges, with respect to graphs with more than sixty thousand transitions.

An important future work will be about computing smaller finite automata $\mathcal{F}$. In fact, often, the full bisimilarity graph is not of interest for the verification purposes, but only a "significant" part of it is needed. The present tool either computes the full bisimilarity graph, or requires a finite automaton to be given manually. In order to simplify the finite automaton, the user often has to work on the full bisimilarity graph, which can be too large to process (and understand). Further tool support is needed to automatically generate some parts of the bisimilarity graph, and to simplify these graphs. This will help the user concentrate on important properties to be verified, but also reduce running time. Moreover, if the timed automaton is not shrinkable with respect to a given finite automaton, the tool could also inform the user about the possibilities of eliminating from the finite automaton those behaviors absent from the shrunk timed automaton. In fact, the counter-examples found by the tool needs to be interpreted by the user, who also decides whether they are important or not.

Part III

Robust Controller Synthesis

This part contains contributions on robust controller synthesis algorithms. In the controller synthesis approach, a given model is no longer seen as a final design; but the goal is to compute a strategy to resolve the non-determinism while satisfying a given objective. The subsequent chapters studies this problem under parameterized perturbation models. Hence, we are interested in synthesizing strategies that are correct even when the moves they prescribe are perturbed by a bounded amount. Chapter 7 considers the problem under the excess perturbation game semantics for reachability objectives on timed automata and turn-based timed games. Chapter 8 considers the conservative perturbation game semantics for Büchi objectives on timed automata. Both semantics are applied to weighted timed automata and games in Chapter 9

Chapter 7

Reachability in Excess Semantics

7.1 Introduction

In this chapter, we study the synthesis of robust controllers in timed automata and games under the excess-perturbation game semantics. The main result of this chapter is the following: We show that deciding the existence of $\delta > 0$, and of a strategy for the controller so as to ensure reachability of a given location in a turn-based timed game (whatever the imprecision, up to δ), is EXPTIME-complete. Moreover, if there is a strategy, we can compute a *uniform* one, which is parameterized by δ, using *shrunk difference bound matrices* (shrunk DBMs) seen in Chapter 5. In this case, our algorithm provides a bound $\delta_0 > 0$ such that the strategy is correct for all $\delta \in [0, \delta_0]$. Our strategies also give quantitative information on how perturbations accumulate or can compensate. Technically, the results require extending shrunk DBMs by *constraints*, and establishes new algebraic properties of this data structure (Chapter 3). The main result is then obtained by transforming the infinite-state game into a finite abstraction, which we prove can be used to symbolically compute a winning strategy, if any (Section 7.5).

The results presented in this chapter were published in [BMS12].

Formally, we are interested in the following problem, called *parameterized robust reachability in the excess perturbation game semantics*.

Definition 7.1.1. *The* parameterized robust reachability in the excess perturbation game semantics *asks, given a turn-based timed game $\mathcal{A}$ and a target location ℓ, whether there exists $\delta > 0$ such that Controller has a winning strategy in $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ for the reachability objective ℓ.*

Notice that we are interested in the parameterized problem: δ is not fixed in advance. When δ is fixed, the problem can be formulated and solved as a usual timed game (see [CHP11] for such an encoding). The parameterized version is important since one does not necessarily have a precise estimation for perturbations at the design phase. It is also interesting to directly synthesize δ and a *uniform* strategy, as described above, rather than finding one by trial-and-error.

Our main result is the EXPTIME-completeness of this problem:

Theorem 7.1.2. *Parameterized robust reachability in the excess perturbation game semantics is* EXPTIME*-complete both for timed automata and turn-based timed games.*

Checking parameterized robust reachability is different from usual reachability checking in the exact semantics mainly for two reasons. First, in order to reach a given location, Controller has to choose the delays along a run, in such a way that uncontrollable perturbations do not accumulate and block the run. In particular, it shouldn't play too close to the borders of the guards (see Fig. 3.3). Second, due to these uncontrollable perturbations, some regions that are not reachable in the absence of perturbation can become reachable (see Fig. 7.1). So, Controller must also be able to win from these newly reachable regions. The regions that become reachable in our semantics are those *neighboring* reachable regions.

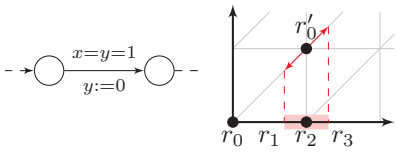


Figure 7.1: Perturbing one transition.

In all timed automata and games we consider, we assume that all clocks are bounded above by a constant. We do not lose generality with this hypothesis since reachability objectives are preserved by choosing a constant large enough, and requiring that all clocks stay below it at any transition.

The chapter is organized as follows. In Section 7.2, we introduce *constraints* on shrunk DBMs, and study *neighborhoods* of shrunk DBMs. For the sake of readability, we first present and prove the algorithm for timed automata, in Section 7.5, then explain how to extend the algorithm to turn-based timed games in Section 7.6. The EXPTIME-hardness result is given in Section 7.7 and holds already for timed automata.

7.2 Shrinking constraints

Consider a transition of a timed automaton, as depicted on Fig. 7.1. From region r_0, the game can reach regions r_1, r_2, r_3, depending on the move of Perturbator. Therefore, in order to win, Controller needs a winning strategy from all three regions. One can then inductively look for winning strategies from these regions; this will generally require shrinking, as exemplified in Fig. 3.3. However, not all shrinkings of these regions provide a winning strategy from r_0. In fact, r_1 (resp. r_3) should not shrink from the right (resp. left) side: their union should include the shaded area, thus points that are arbitrarily close to r_2. In order to define the shrinkings that are useful to us, we introduce shrinking constraints.

Definition 7.2.1. *Let M be a DBM. A* shrinking constraint *for M is a $|\mathcal{C}_0| \times |\mathcal{C}_0|$ matrix over $\{0, \infty\}$. A shrinking matrix P is said to* respect *a shrinking constraint S if $P \leq S$, where the comparison is component-wise. A pair $\langle M, S\rangle$ of a DBM and a shrinking constraint is called a* constrained DBM.

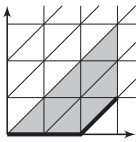
Shrinking constraints specify which facets of a given zone one is (not) allowed to shrink (see Fig. 7.2).

We will focus our attention to *well shrinking constraints* defined below, show that shrinking constraints have a normal form (Subsection 7.2), and then show how shrinking constraints can be propagated in order to deduce new shrinking constraints as in the example of Fig. 7.1 (Subsection 7.2).

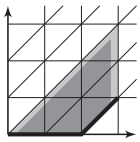
Normalization and Well-Shrinking Constraints.

A shrinking constraint S for a DBM M is said to be *well* if for any SM $P \leq S$, (M, P) is non-empty. A *well constrained DBM* is a constrained DBM given with a well shrinking constraint. We say that a shrinking constraint S for a DBM M is *normalized* if it is the minimum among all equivalent shrinking constraints: for any shrinking constraint S' if for all SMs P, $P \leq S \Leftrightarrow P \leq S'$, then $S \leq S'$. Similarly to the normalization of DBMs or SMs, normalized shrinking constraints contain

$$\begin{array}{c} \\ 0 \\ x \\ y \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & y \end{array} \\ \begin{pmatrix} (0,\leq),0 & (0,<),\infty & (0,<),0 \\ (3,<),\infty & (0,\leq),0 & (2,<),0 \\ (3,<),\infty & (0,<),\infty & (0,\leq),0 \end{pmatrix} \end{array}$$



(a) A constrained DBM $\langle M, S\rangle$ and its representation



(b) A shrinking of $\langle M, S\rangle$

Figure 7.2: Consider a zone defined by $0 < x < 3$, $0 < y < 3$, and $0 < x - y < 2$. Let the shrinking constraint S be defined by $S_{0,y} = 0$, $S_{x,y} = 0$, and $S_{z,z'} = \infty$ for other components. The resulting $\langle M, S\rangle$ is depicted on the left, as a matrix (where, for convenience, we merged both matrices into a single one) and as a constrained zone (where a thick segment is drawn for any boundary that is not "shrinkable", i.e., with $S_{z,z'} = 0$). On the right, the dark gray area represents a shrinking of M that satisfies S.

the tightest constraints implied by the original shrinking constraint. They can be normalized by a procedure that is slightly different from the one used for the normalization of DBMs. This is defined formally in the following lemma.

Lemma 7.2.2. *Let M be a normalized DBM. For any shrinking constraint S for M, there exists a minimum shrinking constraint S' for M such that $S' \leq S$, and for any normalized non-empty shrunk zone (M, P), $P \leq S$ if, and only if, $P \leq S'$.*

Moreover, S' can be obtained as follows. Start with $S' = S$. For every pair $x, y \in \mathcal{C}_0$, if $S_{x,y} = 0$, then set $S'_{z,z'}$ to 0 for all edges (z, z') of all paths in $\Pi_{x,y}(\mathsf{G}(M))$. Also assign $S'_{x,y} = 0$ whenever $\mathsf{M}_{x,y} = \infty$. If $S = S'$, then S is said to be normalized.

Proof. Consider S' obtained from S by the above procedure. Obviously, $S' \leq S$, so that for any SM $P \leq S'$, we have also $P \leq S$. Conversely, consider any normalized non-empty shrunk DBM (M, P) with $P \not\leq S'$, for instance $P_{x,y} \geq 1$ for some x, y where $S'_{x,y} = 0$. If $S_{x,y} = 0$, then clearly, $P_{x,y} \not\leq S_{x,y}$. Otherwise, $S_{x,y} = \infty$ and $S'_{x,y} = 0$ indicate that there exists $z_1, z_2 \in \mathcal{C}_0$ such that $S_{z_1,z_2} = 0$ and there is a path in $\Pi_{z_1,z_2}(\mathsf{G}(M))$ that goes through edge (x, y). Let us denote this path with π. Then since P is normalized, $P_{z_1,z_2} \geq w(\pi)$, and since π contains the edge (x, y), we have $P_{z_1,z_2} \geq P_{x,y} \geq 1$, therefore $P \not\leq S$.

For any pair $x, y \in \mathcal{C}_0$ with $S'_{x,y} = \infty$ let P be the matrix with 0's on all components except for $P_{x,y} = 1$. Let P' be the SM such that $(M, P') = \mathsf{norm}((M, P))$. The normalization procedure can only increase the components so $P'_{x,y} \geq 1$. We have $P' \leq S'$, since for any z_1, z_2 such that $S'_{z_1,z_2} = 0$, (x, y) is not on a path of $\Pi_{z_1,z_2}(\mathsf{G}(M))$. So, for any shrinking constraint S'' such that $S''_{x,y} < S_{x,y}$ for some x, y, there exists a SM P with $P \leq S'$ and $P \not\leq S''$. Therefore, S' is minimal. □

Clearly, the normalization of a shrinking constraint depends on the DBM M, since $\Pi_{x,y}(\mathsf{G}(M))$ depends on $\mathsf{G}(M)$; this is also the case for SMs (Lemma 3.4.3). In the sequel, unless otherwise stated, all shrinking constraints are assumed to be normalized.

The following property of normalized shrinking constraints is easy to prove, using the previous lemma. Given a shrinking constraint S and a SM P, it will allow us to consider that S is normalized, instead of requiring that P is.

Lemma 7.2.3. *Let M be a normalized DBM and S a normalized shrinking constraint. Let P be any SM such that (M, P) is non-empty, and let $(M', P') = \mathsf{norm}((M, P))$. Then, $P \leq S$ if, and only if, $P' \leq S$.*

Proof. Since the normalization procedure for computing P' increases the components, the reverse implication holds. Conversely, assume that $P \leq S$, and pick $x, y \in \mathcal{C}_0$ such that $S_{x,y} = 0$ (for the other entries, there is nothing to be checked on P'). Then $P_{x,y} = 0$. From Lemma 7.2.2, $S_{z,z'} = 0$ for all edges of all paths in $\Pi_{x,y}(\mathsf{G}(M))$, so that also $P_{z,z'} = 0$ for these edges. Applying the definition of $P'_{x,y}$ from Lemma 3.4.3, we get $P'_{x,y} = 0$. □

Propagating Shrinking Constraints.

Lemma 7.2.4 shows that shrinking constraints can be propagated along operations on DBMs. Each item is illustrated in Fig. 7.3; some comments are given after the proof.

Lemma 7.2.4. *Let M, N, O be normalized non-empty DBMs.*

1. *Assume that $M = \mathsf{Pre}_{\mathsf{time}}(N)$. For any normalized well shrinking constraint S for M, there exists a normalized well shrinking constraint S' for N such that for any shrunk DBM (N', Q) with $\mathsf{N}' = \mathsf{N}$, the following holds: $Q \leq S'$ if, and only if, the SM P s.t. $(M', P) = \mathsf{Pre}_{\mathsf{time}}((N', Q))$ satisfies $P \leq S$.*

2. *Assume that $M = N \cap O \neq \emptyset$.*

   (a) *For any normalized well shrinking constraint S for M, there exists a normalized well shrinking constraint S' for N such that for any shrunk DBM (N', Q) with $\mathsf{N}' = \mathsf{N}$ and $N' \cap O \neq \emptyset$, the following holds: $Q \leq S'$ if, and only if the SM P s.t. $(M', P) = (N', Q) \cap O$ satisfies $P \leq S$.*

   (b) *For any normalized well shrinking constraint S for N, there exists a normalized well shrinking constraint S' for M such that for any shrunk DBM (M', Q) with $\mathsf{M} = \mathsf{M}'$ and $M \subseteq M'$, the following holds: $Q \leq S'$ if, and only if there exists an SM $P \leq S$ such that $(N, P) \cap O \subseteq (M', Q)$. Moreover, if $(N, P) \cap O \neq \emptyset$ for all SMs $P \leq S$, then S' is a well shrinking constraint.*

3. *Assume that $M = \mathsf{Unreset}_R(N)$. For any normalized well shrinking constraint S for M, there exists a normalized well shrinking constraint S' for N such that for any SM Q, the following holds: $Q \leq S'$ if, and only if the SM P s.t. $(M', P) = \mathsf{Unreset}_R((N, Q))$ satisfies $P \leq S$.*

All shrinking constraints S' can be computed in polynomial time.

Proof. ▶ **Pretime.** Observe that M is obtained from N by first setting the first row of N to $(0, \leq)$ (write N' for this intermediary DBM) and then applying normalization. Since N is normalized, and we only consider valuations with non-negative values, we have $N_{0,y} \leq 0$ for all $y \in \mathcal{C}$. Consider $(x, y) \in \mathcal{C} \times \mathcal{C}_0$, and a path in $\Pi_{x,y}(N)$. If that path does not visit the state 0, then it has the same weight in N' as in N. Otherwise, its weight in N' is larger than in N, thus it is larger than $N'_{x,y}$. In both cases, $N'_{x,y}$ will not change during the normalization phase. So the normalization of N can only change the first row.

Let $S'_{x,y} = S_{x,y}$ for $(x, y) \in \mathcal{C} \times \mathcal{C}_0$, $S'_{0,y} = \infty$ for $y \in \mathcal{C}$, and $S'_{0,0} = 0$. We show that S' satisfies the desired property. Consider a shrunk DBM (N', Q) with $Q \leq S'$ and $\mathsf{N} = \mathsf{N}'$, and let P be the shrinking matrix such that $(M', P) = \mathsf{Pre}_{\mathsf{time}}((N', Q))$. Let us write $M'' = \mathsf{Pre}_{\mathsf{time}}(N')$. The shrunk DBM (M', P) is obtained by normalizing the shrunk DBM (M'', Q') where Q' is derived from Q by setting the first row to 0 (Lemma 3.4.5).

We show that $P \leq S$. Observe that $\mathsf{M} = \mathsf{M}' = \mathsf{M}''$ so the graphs these DBMs define are the same. First suppose that $S_{x,y} = 0$ for some $x, y \in \mathcal{C} \times \mathcal{C}_0$. For such x, y, we have $\mathsf{N}_{x,y} = \mathsf{M}_{x,y}$, and since $N' \subseteq M''$, we have $\Pi_{x,y}(\mathsf{G}(M)) \subseteq \Pi_{x,y}(\mathsf{G}(N))$. Now, we have $P_{x,y} > 0$ if, and only if there is a path in the former set with positive weight in Q' (Lemma 3.4.3). But this would imply that the same path has positive weight in Q since $Q' \leq Q$, and moreover this path belongs to $\Pi_{x,y}(\mathsf{G}(N))$. This would in turn imply that $Q_{x,y} > 0$, but we have $S_{x,y} = S'_{x,y} = 0$, which contradicts $Q \leq S'$. It follows that $P_{x,y} \leq S_{x,y}$.

Now suppose that $S_{0,y} = 0$ for some $y \in \mathcal{C}$. Since S is assumed to be normalized, along all paths of $\Pi_{0,y}(\mathsf{G}(M))$, all edges (z, z') satisfy $S_{z,z'} = 0$ (Lemma 7.2.2). Since $Q'_{0,z} = 0$ for all z, and that $Q'_{z,z'} = Q_{z,z'} = 0$ for all $z, z' \neq 0$, we get $P_{0,y} = 0$ (Lemma 3.4.3). Hence $P \leq S$.

We now show that for any SMs P and Q s.t. $Q \not\leq S'$ and $(M', P) = \mathsf{Pre}_{\mathsf{time}}((N', Q))$, it holds $P \not\leq S$. Consider any SM $Q \not\leq S'$ such that (w.l.o.g.) (N', Q) is normalized. Then $S'_{x,y} = 0$ and $Q_{x,y} \geq 1$ for some x, y. By construction of S', we have $x \neq 0$, and $S_{x,y} = 0$. Moreover, since $x \neq 0$, we have $\mathsf{M}_{x,y} = \mathsf{N}_{x,y}$, and $\mathsf{N}_{x,y} < \infty$ since $Q_{x,y} \geq 1$ and (N', Q) is normalized. Let P denote the normalized SM such that $(M', P) = \mathsf{Pre}_{\mathsf{time}}((N'', Q))$. P is obtained by letting the first row of Q to zero and applying normalization. We get $P_{x,y} \geq Q_{x,y} \geq 1$, therefore $P \not\leq S$.

To show that S' is a well shrinking constraint, assume that for some $Q \leq S'$, (N', Q) is empty. By definition of S', there exists $P \leq S$ such that $(M', P) = \mathsf{Pre}_{\mathsf{time}}((N', Q))$, so (M', P) is also empty, which contradicts the fact that S is a well shrinking constraint for M.

► **Intersection (a).** For any SMs P, Q, we have $(M, P) = (N, Q) \cap O$ if, and only if $(M', P) = (N, Q) \cap M$ for some $\mathsf{M}' = \mathsf{M}$. So it suffices to consider the equation $M = N \cap M$ with $M \subseteq N$. Moreover, notice that $\emptyset \neq M = M \cap N'$ for any $N \subseteq N'$.

We let $S'_{x,y} = S_{x,y}$ for any $x, y \in \mathcal{C}_0$ such that $\mathsf{M}_{x,y} = \mathsf{N}_{x,y}$ and $S'_{x,y} = \infty$ otherwise, and make S' normalized. Observe that the weight of any path is smaller or equal in $\mathsf{G}(M)$ than in $\mathsf{G}(N)$, since $M \subseteq N$ and both DBMs are normalized. Consider any shrunk DBM (N', Q) with $\mathsf{N} = \mathsf{N}'$, $N \subseteq N'$ and $Q \leq S'$ and let P such that $(M', P) = (N', Q) \cap M$. Let us write $M'' = N' \cap M$. Again, $\mathsf{G}(M) = \mathsf{G}(M') = \mathsf{G}(M'')$. Here, P is obtained from Q by first defining Q' as $Q'_{x,y} = 0$ if $\mathsf{M}_{x,y} < \mathsf{N}_{x,y}$ and $Q'_{x,y} = Q_{x,y}$ otherwise. Then P is the normalization of Q'. We show, by induction on $n \geq 2$, that for any pair $x, y \in \mathcal{C}_0$ with $S_{x,y} = 0$, any path of $\Pi_{x,y}(\mathsf{G}(M))$ visiting at most n states have weight 0 in Q'. This entails that $P_{x,y} = 0$ whenever $S_{x,y} = 0$.

- When $n = 2$, if $\mathsf{M}_{x,y} = \mathsf{N}_{x,y}$, then $S'_{x,y} = 0$ by definition of S', so $Q_{x,y} = 0$, and $Q'_{x,y} = 0$. If $\mathsf{M}_{x,y} < \mathsf{N}_{x,y}$, we have $Q'_{x,y} = 0$ by the definition of Q'.
- Consider $n \geq 3$. Let $\pi = x_1 x_2 \dots x_n$ be a path in $\Pi_{x,y}(\mathsf{G}(M))$. Since $S_{x,y} = S_{x_1,x_n} = 0$, we also have $S_{x_1,x_2} = 0$ and $S_{x_2,x_n} = 0$ By induction, $Q'_{x_1,x_2} = 0$ and all paths of $\Pi_{x_2,x_n}(\mathsf{G}(M))$ of length at most $n-1$ have weight 0 in Q'. Hence, π has weight 0 in Q'.

Assume now $Q \not\leq S'$, i.e., $Q_{x,y} \geq 1$ and $S'_{x,y} = 0$ for some $x, y \in \mathcal{C}_0$. Then we must have $\mathsf{M}_{x,y} = \mathsf{N}_{x,y}$ and $S_{x,y} = 0$. Let $(M', P) = (N', Q) \cap M$. We know that P is the normalization of Q', and that $Q'_{x,y} = Q_{x,y} \geq 1$. So $P_{x,y} \geq Q'_{x,y} \geq 1$. Hence $P_{x,y} \not\leq S_{x,y}$.

We have that S' is a well shrinking constraint for N, since otherwise this would imply that S is not a well shrinking constraint for M, as in the previous case.

► **Intersection (b).** Using the same argument as in the previous case, we can assume that $O = M \subseteq N$. Since M and N are normalized and non-empty, we have $\mathsf{M}_{x,y} \leq \mathsf{N}_{x,y}$ for all $x, y \in \mathcal{C}_0$. We define S'_1 by $S'_{1\,x,y} = S_{x,y}$ if $\mathsf{M}_{x,y} = \mathsf{N}_{x,y}$ and $S'_{1\,x,y} = 0$ otherwise. Let S' be defined as $S'_{x,y} = \max_{\pi \in \Pi_{x,y}(\mathsf{G}(M))} S'_1(\pi)$.

Consider any shrunk DBM (M', Q) with $\mathsf{M}' = \mathsf{M}$, $(M', Q) \subseteq M$, and $Q \leq S'$. Let us show that there is (N, P) with $P \leq S$ with the desired property. Let us first define Q' as follows:

$$Q'_{x,y} = \begin{cases} 0 & \text{if } \mathsf{M}_{x,y} < \mathsf{N}_{x,y} \text{ or } S_{x,y} = 0 \\ \max_{z,z' : (x,y) \in \Pi_{z,z'}(\mathsf{G}(M))} Q_{z,z'} & \text{if } \mathsf{M}_{x,y} = \mathsf{N}_{x,y} \text{ and } S_{x,y} = \infty. \end{cases}$$

Let us write $(N', P) = \mathsf{norm}((N, Q'))$. If $S_{x,y} = 0$ for some $x, y \in \mathcal{C}_0$, then along all edges $(z, z') \in \Pi_{x,y}(\mathsf{G}(N))$, we have $S_{z,z'} = 0$, therefore $Q'_{z,z'} = 0$. Thus, $P_{x,y} = 0$ (from Lemma 3.4.3) and we get $P \leq S$.

Let (M'', Q'') denote the shrunk DBM defined by $(M'', Q'') = (N', P) \cap M$ with $\mathsf{M}'' = \mathsf{M}$ (Lemma 3.4.5). Let us show that $Q \leq Q''$, which implies, together with $M'' \subseteq M \subseteq M'$ that $(M'', Q'') \subseteq (M', Q)$ as desired. Q'' is the normalization of the SM P' defined as follows: $P'_{x,y} = P_{x,y}$ if $\mathsf{M}_{x,y} = \mathsf{N}_{x,y}$ and $P'_{x,y} = 0$ otherwise. Let $x, y \in \mathcal{C}_0$.

- Assume that $\mathsf{M}_{x,y} = \mathsf{N}_{x,y}$. If $S'_{x,y} = 0$, then $Q_{x,y} = 0$ so clearly $Q''_{x,y} \geq Q_{x,y}$. Otherwise, there exists a path $\pi \in \Pi_{x,y}(\mathsf{G}(M))$ such that for some edge $(z, z') \in \pi$, $\mathsf{M}_{z,z'} = \mathsf{N}_{z,z'}$ and $S_{z,z'} = \infty$. By definition of Q', we have $Q'_{z,z'} \geq Q_{x,y}$. Moreover $P'_{z,z'} = P_{z,z'} \geq Q_{z,z'}$. The normalization of P' then yields $Q''_{x,y} \geq P'_{z,z'} = P_{z,z'} \geq Q_{z,z'} \geq Q_{x,y}$.
- Otherwise, $\mathsf{M}_{x,y} < \mathsf{N}_{x,y}$. If $S'_{x,y} = 0$, the result is clear. If $S'_{x,y} = \infty$, then there must be a path in $\Pi_{x,y}(\mathsf{G}(M))$ with an edge (z, z') with $\mathsf{M}_{z,z'} = \mathsf{N}_{z,z'}$ and $S_{z,z'} = \infty$. Then, $Q'_{z,z'} \geq Q_{x,y}$. So the normalization of P' yields $Q''_{x,y} \geq P'_{z,z'} = P_{z,z'} \geq Q'_{z,z'} \geq Q_{x,y}$.

We now show the converse direction. Consider any $x, y \in \mathcal{C}_0$ such that $S'_{x,y} = 0$. Let us show that for any (N, P) with $P \leq S$, if (M', Q'') denotes the shrunk DBM such that $(M', Q'') = M \cap (N, P)$, then $Q''_{x,y} = 0$. This proves that if $Q_{x,y} \geq 1$ for some SM Q, then there is no matching $P \leq S$ that satisfies the property.

We show, by induction on $n \geq 2$, that if $S'_{x,y} = 0$, then all paths of $\Pi_{x,y}(\mathsf{G}(M))$ of length at most n have weight zero in P'.

- Let $n = 2$. If $\mathsf{M}_{x,y} = \mathsf{N}_{x,y}$, then $P'_{x,y} = P_{x,y}$. But we have $S_{x,y} = 0$, so $P_{x,y} = 0$. If $\mathsf{M}_{x,y} < \mathsf{N}_{x,y}$, then $P'_{x,y} = 0$ by definition.
- If $n \geq 3$, consider any path $\pi = x_1 x_2 \dots x_n$ in $\Pi_{x,y}(\mathsf{G}(M))$. Suppose that $\mathsf{M}_{x_j,x_{j+1}} = \mathsf{N}_{x_j,x_{j+1}}$ for all $1 \leq j \leq n-1$. Then π is also a shortest path in $\mathsf{G}(N)$, hence $\mathsf{M}_{x,y} = \mathsf{N}_{x,y}$, and $S_{x_j,x_{j+1}} = 0$ for all $1 \leq j \leq n-1$, as S is normalized. It follows that $P_{x_j,x_{j+1}} = 0$, hence also $P'_{x_j,x_{j+1}} = 0$.

  Suppose now that for some $1 \leq j \leq n-1$, $\mathsf{M}_{x_j,x_{j+1}} < \mathsf{N}_{x_j,x_{j+1}}$. We have $P'_{x_j,x_{j+1}} = 0$. On the other hand, $S'_{x_1,x_j} = S'_{x_{j+1},x_n} = 0$ since otherwise we would get $S'_{x,y} = \infty$. By induction, all paths of length at most $n-1$ in $\Pi_{x_1,x_j}(\mathsf{G}(M))$ and $\Pi_{x_{j+1},x_n}(\mathsf{G}(M))$ have weight 0 in P'. So π has weight 0 in P'.

We now show that S' is a well shrinking constraint for M, given the hypothesis that $(N, P) \cap M \neq \emptyset$ for all SMs $P \leq S$. But this immediately implies that S' is a well shrinking constraint, since for any $Q \leq S'$ there exists $P \leq S$ with $\emptyset \subsetneq (N, P) \cap M \subseteq (M, Q)$.

► **Unreset.** Let N_R denote the DBM that defines the (largest) zone satisfying $\wedge_{x \in R} x = 0$. Since we have $M = \mathsf{Unreset}_R(N \cap N_R)$, by the previous case, we may assume that $N \subseteq N_R$. The DBM M is obtained from N by replacing each component (x, y) with $x \in R$ by ∞. For any $y \in \mathcal{C}_0$, we

define $S''_{x,y} = S_{x,y}$ if $x \in \mathcal{C} \setminus R$, and $S''_{0,y} = 0$, and $S''_{x,y} = \infty$ if $x \in R$. Then S' is obtained by normalizing S''.

Let Q be any normalized SM with $Q \leq S'$, and P be the (normalized) SM such that $(M', P) = \mathsf{Unreset}_R((N, Q))$. Let Q' denote the SM obtained from Q by setting each component (x, y), with $x \in R$, to zero. Then, P is the normalization of Q'. Let us show that $P \leq S$. Consider any $x, y \in \mathcal{C}_0$ with $S_{x,y} = 0$.

- If $x \in \mathcal{C} \setminus R$ then $S'_{x,y} = S_{x,y} = 0$ and $Q_{x,y} = 0$. So, along all paths $\Pi_{x,y}(\mathsf{G}(N))$, S' is zero, and so are the weights in Q, and also the weights in Q'. Now, clearly, $\Pi_{x,y}(\mathsf{G}(M)) \subseteq \Pi_{x,y}(\mathsf{G}(N))$. So after normalization of Q', we have $P_{x,y} = 0$.
- If $x = 0$, then $S'_{x,y} = 0$ and $Q_{x,y} = 0$. The argument is then similar.
- If $x \in R$, then $M_{x,y} = \infty$. Thus $\Pi_{x,y}(\mathsf{G}(M)) = \emptyset$, so the normalization of Q' yields $P_{x,y} = 0$.

Consider a normalized SM Q with $Q \not\leq S'$. We must have $Q_{x,y} \geq 1$ and $S''_{x,y} = 0$ for some $x, y \in \mathcal{C}_0$, by Lemma 7.2.2. The latter condition entails that $x \notin R$. Let Q' denote the DBM obtained from Q by replacing all components (x, y) where $x \in R$ with 0. Then the normalization of Q' yields the SM P such that $(M', P) = \mathsf{Unreset}_R((N, Q))$. Since $x \in \mathcal{C}_0 \setminus R$, then $S_{x,y} = S'_{x,y} = 0$ and $Q'_{x,y} = Q_{x,y} \geq 1$. The normalization of Q' can only increase its components, so $P_{x,y} \geq 1$, hence $P \not\leq S$.

We show that S' is a well shrinking constraint for N as in the previous cases. □

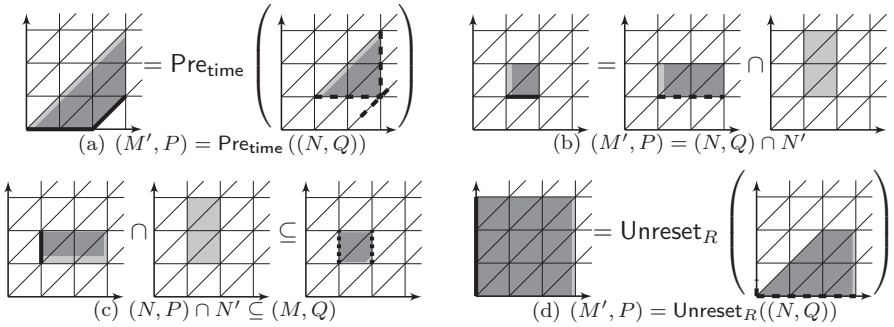


Figure 7.3: Each of the figures illustrates one item in Lemma 7.2.4. In each case, DBMs M, N and N' are fixed and satisfy the "unshrunk" equation (that is, when $P = Q = \mathbf{0}$). The thick plain segments represent the fixed shrinking constraint S. The thick dashed segments represent the shrinking constraint S' that is constructed. In each case, if a SM Q is chosen, it holds that $Q \leq S'$ iff there is an SM $P \leq S$ that satisfies the equation. For instance, if $Q_{x,0} \geq 1$ in (a), then the DBM describing the time predecessors of (N, Q) shrinks the diagonal component (x, y), which violates S.

We now extend the previous lemma to the operator shrink^+. In this case, the existence of a shrinking constraint depends on the given constrained DBM. The lemma is illustrated in Fig. 7.4.

Lemma 7.2.5. *Let M be a normalized non-empty DBM and S be a normalized shrinking constraint.*

- *If $M_{x,0} < \infty$ and $S_{x,0} = 0$ for some $x \in \mathcal{C}$, then for all SMs Q, the shrunk DBM $(M', P) = \mathsf{shrink}^+((M, Q))$ does not satisfy $P \leq S$.*

- *Otherwise, for all SMs Q, the following holds: $Q \leq S$ if, and only if, the shrunk DBM $(M', P) = \mathsf{shrink}^+((M, Q))$ satisfies $P \leq S$.*

Proof. If $S_{x,0} = 0$ and $\mathsf{M}_{x,0} < \infty$ for some $x \in \mathcal{C}$, since shrink^+ increases each (but one) component of the first column of the SM by one, and since the normalization can only increase components for which $\mathsf{M}_{x,0} < \infty$, there is no $P \leq S$ such that $(M', P) = \mathsf{shrink}^+((M, Q))$.

Assume that $S_{x,0} = \infty$ for all $x \in \mathcal{C}$ with $\mathsf{M}_{x,0} < \infty$. Consider any $Q \not\leq S$. The operator shrink^+ increments all components $(x, 0)$ of the SM by one, and applies normalization. So if $Q_{x,y} > S_{x,y} = 0$ for some $x, y \in \mathcal{C}$, we know that if P denotes the SM such that $(M', P) = \mathsf{shrink}^+((M, Q))$, then $P_{x,y} \geq Q_{x,y} > 0$, since shrink^+ only increases the components of the SM. Thus, for SMs $Q \not\leq S$, there is no corresponding $P \leq S$ with the desired property. Consider now any $Q \leq S$, and P such that $(M', P) = \mathsf{shrink}^+((M, Q))$. Assume that $P_{x,y} \geq 1$ for some $S_{x,y} = 0$. We have $\mathsf{M}_{x,y} < \infty$ since (M', P) is normalized. Since $Q_{x,y} = 0$, there exists a path in $\Pi_{x,y}(\mathsf{G}(M))$ that contains an edge $(z, 0)$ such that $Q_{z,0} = 0$, and $P_{z,0} \geq 1$. But then $S_{z,0} = 0$, which contradicts our assumption. □

(a) $(M', P) = \mathsf{shrink}^+((M, Q))$ but $P \not\leq S$ (b) $(M', P) = \mathsf{shrink}^+((M, Q))$ with $P \leq S$.

Figure 7.4: Examples for both cases of Lemma 7.2.5

Let us comment on Fig. 7.3, the figure about $\mathsf{Pre}_{\mathsf{time}}()$, and how it can be used for our purpose. Assume there is an edge guarded by N (the whole gray area in the right) without resets. In the non-robust setting, this guard can be reached from any point of M (the whole gray area in the left). If we have a shrinking constraint S on M, and we want to synthesize a winning strategy from a shrinking of M satisfying S, then Lemma 7.2.4 gives the shrinking constraint S' for N, with the following property: given any shrinking (N, Q), we can find $P \leq S$ with $(M, P) = \mathsf{Pre}_{\mathsf{time}}((N, Q))$ (hence, we can delay into (N, Q)), if, and only if Q satisfies $Q \leq S'$. The problem is now "reduced" to finding a winning strategy from $\langle N, S' \rangle$. However, forward-propagating these shrinking constraints is not always that easy. We also need to deal with resets, with the fact that Controller has to choose a delay greater than $\delta > 0$, and also with the case where there are several edges leaving a location. This is the aim of the following developments.

7.3 Neighborhoods

We now consider constrained regions, which are constrained DBMs in which the DBM represents a region. Fig. 7.1 shows that if Controller plays to a region, then Perturbator can reach some of the surrounding regions, shown by the arrows. To characterize these, we define the set of *neighboring regions* of $\langle r, S \rangle$ as,

$$\mathcal{N}_{r,S} = \left\{ s \;\middle|\; s \lessdot^* r \text{ or } r \lessdot^+ s, \text{ and } \forall Q \leq S.\ s \cap \mathsf{enlarge}((r, Q)) \neq \emptyset \right\}$$

where $\mathsf{enlarge}((r,Q))$ is the shrunk DBM (M,P) such that $v+[-\delta,\delta] \subseteq M-\delta P$ for every $v \in r-\delta Q$. This is the set of regions that have "distance" at most δ to any shrinking of the constrained region (r,S). We write $\mathsf{neighbor}\langle r,S\rangle = \bigcup_{s\in\mathcal{N}_{r,S}} s$.

Lemma 7.3.1 (Neighborhood)**.** *Let $\langle r,S\rangle$ be a well constrained region. Then $\mathsf{neighbor}\langle r,S\rangle$ is a zone. If N is the corresponding normalized DBM, there exists a well shrinking constraint S' such that for every SM Q, $Q \le S'$ iff the SM P defined by $(r',P) = r \cap \mathsf{shrink}_{[-\delta,\delta]}((N,Q))$, satisfies $P \le S$. The pair $\langle N,S'\rangle$ is the* constrained neighborhood *of $\langle r,S\rangle$, and it can be computed in polynomial time. Moreover, the constraint S' is such that $S'_{x,y} = S_{x,y}$ for every $x,y \in \mathcal{C}$, and $S'_{x,0} = S'_{0,x} = \infty$ for every $x \in \mathcal{C}$.*

Constrained neighborhoods are illustrated in Fig. 7.5.

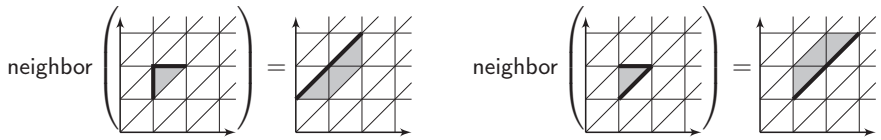


Figure 7.5: Constrained neighborhood of two constrained regions. Notice that inside any shrinking of the constrained region, there is always a valuation such that a perturbation of $[-\delta,\delta]$ moves the valuation to any region of the neighborhood.

To prove Lemma 7.3.1 which states the properties of the neighborhoods of constrained regions, we first need to introduce some notations and simple facts about constrained regions.

Lemma 7.3.2. *Let $\langle r,S\rangle$ be a constrained region and consider the partition $X_1,\ldots,X_n$ of clocks according to their fractional values. Then, for all $x,y \in X_i$ for any $1 \le i \le n$, we have $S_{x,y} = 0$, and for all $x \in X_1$, we have $S_{x,0} = S_{0,x} = 0$. Moreover,*

- *There exists $2 \le i_0 \le n+1$ such that for any $x \in X_i$ with $i_0 \le i \le n$, $S_{x,0} = 0$, for any $y \in X_{i'}$ with $i \le i' \le n$, $S_{x,y} = 0$, and for any $x \in X_1 \cup \ldots \cup X_{i_0-1}$, $S_{x,0} = \infty$.*
- *There exists $1 \le j_0 \le n$ such that for any $x \in X_j$ with $1 \le j \le j_0$, $S_{0,x} = 0$, for any $y \in X_{j'}$ with $1 \le j' < j$, $S_{y,x} = 0$, and for any $x \in X_j$ with $j_0+1 \le j \le n$, $S_{0,x} = \infty$.*

Proof. This follows from the normalization of S. In fact, suppose $S_{x,0} = 0$ for some $x \in X_i$ with $i \ge 2$, and consider $y \in X_j$ for any $i < j \le n$. Then $(x,y,0)$ is a shortest path in $G(r)$. Therefore, we must have $S_{x,y} = S_{y,0} = 0$. One can then choose i_0 as minimal to satisfy this statement, and j_0 maximal. □

Let us make some remarks about the graphs of regions and their shrinking constraints. Consider a region r. For any $x \in X_i$, $y \in X_j$ and $z \in X_k$ with $i < j < k$, (x,y,z) and (z,x,y) are shortest paths in $G(r)$. In fact, $r_{x,z} = r_{x,y} + r_{y,z}$ since $r_{x,y} = r_{x,0} - r_{y,0}$, $r_{y,z} = r_{y,0} - r_{z,0}$, and $r_{x,z} = r_{x,0} - r_{z,0}$. Similarly, $r_{z,x} = r_{z,0} - r_{x,0} + 1$ and $r_{z,y} = r_{z,0} - r_{y,0} + 1$ yield $r_{z,y} = r_{z,x} + r_{x,y}$. A shrinking constraint S with $S_{x,y} = 0$ means that a shrinking of r satisfying S, contains valuations ν with $\mathsf{frac}(\nu(y)) - \mathsf{frac}(\nu(x)) \le \epsilon$ for any $\epsilon > 0$. If $S_{y,x} = 0$, this means that some valuations ν satisfy $\mathsf{frac}(\nu(y)) - \mathsf{frac}(\nu(x)) \ge 1 - \epsilon$.

Proof of Lemma 7.3.1. Let us prove that the neighborhood is a zone, and show how to compute it. We characterize the successor regions of r that belong to the neighborhood of $\langle r, S\rangle$. The predecessor regions can be charaterized similarly. Consider the partition of clocks in r ordered according to their fractional parts:

$$0 = \mathsf{frac}(X_1) < \mathsf{frac}(X_2) < \ldots < \mathsf{frac}(X_m) < 1,$$

where X_i's are subsets of clocks having the same fractional part, and X_1 is possibly empty.

- First, assume that $S_{x,0} = \infty$ for $x \in X_m$. Consider the case $X_1 = \emptyset$. Consider any SM $Q \leq S$ such that $Q_{x,0} \geq 2$ for all clocks $x \in X_2 \cup \ldots X_m$. Then for all $\nu \in (r - \delta Q)$, we have $\mathsf{frac}(\nu(x)) \leq 1 - 2\delta$ for all clocks x. We have, for any $t \in [0, \delta]$, $\mathsf{frac}(\nu(x) + t) < 1$. So no region s with $r \lessdot s$ is included in the neighborhood. If $S_{x,0} = \infty$ for $x \in X_m$ but $X_1 \neq \emptyset$, then necessarily $S_{x,0} = S_{0,x} = 0$ for $x \in X_1$ since S is a well shrinking constraint. Then, the neighborhood contains the region s with $r \lessdot s$ but no successor of s, which is shown as above.

- Assume that $S_{x,0} = 0$ for $x \in X_m$. Let $i \in \{1, \ldots, m\}$ be minimum such that X_i is non-empty and $S_{x,0} = 0$ for all $x \in X_j$ and $i \leq j \leq m$. [1] Then, for any $1 \leq j \leq i-1$ and $x \in X_j$ $S_{x,0} = \infty$ by Lemma 7.3.2. Let s be the unique region that satisfies $r \lessdot^+ s$, $s_{x,0} = r_{x,0} + 1$ and $\prec^s_{x,0} = <$ for all $x \in X_i$, and $s_{x,0} = r_{x,0}$ for all $x \in X_{i-1}$. We show that the neighborhood of $\langle r, S\rangle$ contains all regions t such that $r \lessdot^* t \lessdot^* s$, but no region t such that $s \lessdot^+ t$. Let (r, Q, δ_0) be a well-shrinking and consider any $\delta \in [0, \delta_0)$. For any $x \in X_j$ for $\max(2, i) \leq j \leq m$, we have $r_{x,0} - \delta Q_{x,0} = r_{x,0}$, and $-r_{0,x} + \delta Q_{0,x} < r_{x,0}$ (in fact $\prec^r_{x,0} = <$ for these clocks). Let $\epsilon < \min(\delta/2, r_{x,0} - (-r_{0,x} + \delta Q_{0,x}))$ for all $x \in X_j$ and $\max(2, i) \leq j \leq m$, so that $-r_{0,x} + \delta Q_{0,x} < r_{x,0} - \epsilon < r_{x,0}$. By Lemma 3.3.3, there exists a valuation $\nu \in (r - \delta Q)$ such that $r_{x,0} - \epsilon < \nu(x) < r_{x,0}$ for $x \in X_{\max(2,i)}$. Define $d_j = 1 - \mathsf{frac}(\nu(x))$ for $x \in X_j$ for all $i \leq j \leq m$. We know by the fractional ordering that $0 < d_m < d_{m-1} < \ldots < d_i < \delta/2$. Define also $d'_j = \frac{d_j + d_{j-1}}{2}$ for all $i + 1 \leq j \leq m$, and $d_{m+1} = \frac{d_m}{2}$. If $i = 1$, then we have

  $$\mathsf{reg}(\nu) \lessdot \mathsf{reg}(\nu + d_{m+1}) \lessdot \mathsf{reg}(\nu + d_m) \lessdot \mathsf{reg}(\nu + d'_m) \lessdot \mathsf{reg}(\nu + d_{m-1}) \lessdot \ldots \lessdot \mathsf{reg}(\nu + d_i),$$

  and for $\epsilon' > 0$ small enough, $\mathsf{reg}(\nu + d_i) \lessdot \mathsf{reg}(\nu + d_i + \epsilon') = s$. If $i > 1$, we remove $\mathsf{reg}(\nu + d_{m+1})$ since this is equal to $\mathsf{reg}(\nu)$. Let us show now that the time-successor of s is not included in the neighborhood. In fact, if $i = 1$, then, the immediate time successor of s is the unique region $r \lessdot^+ t$ such that $-t_{0,x} = t_{x,0} = r_{x,0} + 1$ and $\prec^t_{x,0} = \leq$ for $x \in X_1$. But for all $\nu \in (r - \delta Q)$, we have $\nu(x) = r_{x,0}$, so $\nu(x) + \delta < t_{x,0}$. If $i = 2$ and $X_1 = \emptyset$, then the immediate time successor t of s satisfies $-t_{0,x} = t_{x,0} = r_{x,0} + 1$ for $x \in X_m$. But since for any $\nu \in r$, $\nu(x) + \delta < t_{x,0}$, t is not in the neighborhood. If $i \geq 2$ and $X_{i-1} \neq \emptyset$, then the proof is similar since the immediate time successor of s is the region where clocks in X_{i-1} become integer. The case of the time-predecessors of r is treated similarly.

From the proof above we can directly define the DBM N that represents the neighborhood of r, as follows. N has the same diagonal constraints as r, so $N_{x,y} = r_{x,y}$ for all $x, y \in \mathcal{C}$. For all $x \in X_1$, we let $\mathsf{N}_{x,0} = \mathsf{r}_{x,0} + 1$ and $\prec^N_{x,0} = <$, and if $\mathsf{r}_{0,x} < 0$, we let $\mathsf{N}_{0,x} = \mathsf{r}_{0,x} + 1$ and $\prec^N_{0,x} = <$, otherwise $\mathsf{N}_{0,x} = \mathsf{r}_{0,x} = 0$ and $\prec^N_{0,x} = \leq$. If $S_{x,0} = \infty$ for all $x \in X_2 \cup \ldots \cup X_m$, then $\mathsf{r}_{x,0} = \mathsf{N}_{x,0}$ for all x. Otherwise, let $i \in \{2, \ldots, m\}$ minimum such that $S_{x,0} = 0$ for all $x \in X_j$ for all $i \leq j \leq m$.

[1] Here, the non-emptiness hypothesis in only significant for $i = 1$.

We let $\mathsf{N}_{x,0} = \mathsf{r}_{x,0} + 1$ and $\prec^N_{x,0} = <$ for all $x \in X_i \cup X_{i+1} \cup \ldots \cup X_m$, and $\mathsf{N}_{x,0} = \mathsf{r}_{x,0}$, $\prec^N_{x,0} = \prec^r_{x,0}$ for all $x \in X_2 \cup \ldots \cup X_{i-1}$. Symmetrically, if $S_{0,x} = \infty$ for all $x \in X_2 \cup \ldots \cup X_m$, we let $\mathsf{N}_{0,x} = \mathsf{r}_{0,x}$ and $\prec^r_{0,x} = \prec^N_{0,x}$. Otherwise, let $i' \in \{2 \ldots, m\}$ be maximum such that $S_{0,x} = 0$ for all $x \in X_j$ for all $2 \leq j \leq i'$. For all $x \in X_j$ and $1 \leq j \leq i'$, if $\mathsf{r}_{0,x} < 0$, we let $\mathsf{N}_{0,x} = \mathsf{r}_{0,x} + 1$ and $\prec^N_{0,x} = <$, and otherwise $\mathsf{N}_{0,x} = 0$ and $\prec^N_{0,x} = \leq$. We let $\mathsf{N}_{0,x} = \mathsf{r}_{0,x}$, $\prec^N_{0,x} = \prec^r_{0,x}$ for all $x \in X_{i'+1} \cup \ldots \cup X_m$.

Let S' be the shrinking constraint given by Lemma 7.2.4, for which for all SMs Q, $Q \leq S'$ if, and only if there exists $P \leq S$ with $(r', P) = r \cap (N, Q)$ with $\mathsf{r}' = \mathsf{r}$. From the proof of this lemma, S' is the normalization of the following shrinking constraint: we let $S'_{x,y} = S_{x,y}$ for all $x, y \in \mathcal{C}$, since $\mathsf{r}_{x,y} = \mathsf{N}_{x,y}$. For all $x \in \mathcal{C}$, $S'_{x,0} = S_{x,0}$ if $\mathsf{r}_{x,0} = \mathsf{N}_{x,0}$ and $S'_{x,0} = \infty$ otherwise. But by construction of N, $\mathsf{r}_{x,0} = \mathsf{N}_{x,0}$ only if $S'_{x,0} = \infty$ so we get $S'_{x,0} = \infty$ for all clocks. Similarly, we get $S'_{0,x} = \infty$ for all $x \in \mathcal{C}$. We are going to show that S' is already normalized. Consider any $x \in \mathcal{C}$. To get a contradiction, suppose that there exists $z, z' \in \mathcal{C}$ with $S'_{z,z'} = S_{z,z'} = 0$ and a path $x_1 \ldots x_n$ in $\Pi_{z,z'}(\mathsf{G}(N))$ that contains the edge $(x, 0)$ – which would imply that $S'_{x,0}$ becomes 0. Since $\mathsf{r}_{z,z'} = \mathsf{N}_{z,z'}$, we have $\Pi_{z,z'}(\mathsf{G}(N)) \subseteq \Pi_{z,z'}(\mathsf{G}(r))$. Then, $S_{x,0} = 0$ since S is normalized. But this means $\mathsf{N}_{x,0} = \mathsf{r}_{x,0} + 1$ by definition of N, and this path cannot be a shortest path in $\mathsf{G}(N)$ (in fact, $\mathsf{N}_{z,z'} = \mathsf{r}_{z,z'} = \sum_{k=1}^{n-1} r_{x_i,x_{i+1}}$). Consider now $x, y \in \mathcal{C}$ such that $S_{x,y} = \infty$ and let us show that $S'_{x,y} = \infty$ after normalization. Suppose there exists a path $\pi \in \Pi_{z,z'}(\mathsf{G}(N))$ where $z, z' \neq 0$, $S_{z,z'} = S'_{z,z'} = 0$ and the edge (x, y) belongs to π. If 0 does not appear in π, then this path also belongs to $\Pi_{z,z'}(\mathsf{G}(r))$, since $\mathsf{r}_{z,z'} = \mathsf{N}_{z,z'}$ and all edges have the same weight in both graphs, so we would have $S_{x,y} = 0$. So, assume that some edges $(w, 0)$ and $(0, w')$ belong to π. If $S_{w,0} = S_{0,w} = \infty$, then $\mathsf{r}_{w,0} = \mathsf{N}_{w,0}$ and $\mathsf{r}_{0,w} = \mathsf{N}_{0,w}$ by definition of N, so π is a shortest path in $\mathsf{G}(r)$, which contradicts $S_{x,y} = \infty$. But, if $S_{w,0} = 0$ or $S_{0,w} = 0$ then $\mathsf{N}_{w,0} = \mathsf{r}_{w,0} + 1$ or $\mathsf{N}_{0,w} = \mathsf{r}_{0,w} + 1$, and in this case π is not a shortest path in $\mathsf{G}(N)$ since $\mathsf{N}_{z,z'} = \mathsf{r}_{z,z'}$. Hence, there is no such path π, and we get $S'_{x,y} = \infty$ after normalization.

Now, let Q' be such that $(N', Q') = \mathsf{shrink}_{[-\delta,\delta]}((N, Q))$. If Q'' denotes the SM obtained from Q' by incrementing by one each component of the first row and the first column (but the $(0, 0)$), then $(N', Q') = \mathsf{norm}((N, Q''))$. Moreover, because $S'_{x,0} = S'_{0,x} = \infty$ for all $x \in \mathcal{C}$, we have $Q', Q'' \leq S''$. Then, by definition of S', there exists $P \leq S$ such that $(r', P) = r \cap (N, Q')$. Conversely, if $Q \not\leq S'$, then $Q' \not\leq S'$. So, if there exists P such that $(r', P) = r \cap (N, Q')$, then $P \not\leq S$, by definition of S'.

The fact that S' is a well shrinking constraint for N follows from the fact that S is a well shrinking constraint for r. In fact, for any $Q \leq S'$, there exists $P \leq S$ such that $(r', P) = r \cap \mathsf{shrink}_{[-\delta,\delta]}((N, Q))$ and (r', P) is non-empty by hypothesis, so (N, Q) cannot be empty. □

7.4 Controllable Predecessors

The following lemma characterizes, given a constrained region $\langle r, S\rangle$, the set of constrained regions $\langle s, S_s\rangle$ such that any shrunk region satisying $\langle s, S_s\rangle$ can be reached by delaying from some shrunk region satisfying $\langle r, S\rangle$. These are the sets whose reachability can be ensured by Controller by a delay.

Lemma 7.4.1. *Let $\langle r, S\rangle$ be a well constrained region, and s be a region such that $r \lessdot^* s$. Then the following properties are equivalent:*

1. *there exists a well shrinking constraint S' (which can be computed in polynomial time) such that for every SM Q, $Q \leq S'$ iff the SM P such that $(r', P) = r \cap \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}((s, Q)))$ for some $\mathsf{r}' = \mathsf{r}$ satisfies $P \leq S$;*

2. $\mathit{neighbor}\langle r, S\rangle \subseteq \mathit{Pre}_{\mathit{time}}(s)$;

3. *define* $N = \mathit{Pre}_{\mathit{time}}(s)$, *and* S_N *such that for all SM* Q, $Q \leq S_N$ *iff the SM* P *defined by* $(r', P) = r \cap (N, Q)$ *with* $\mathsf{r}' = \mathsf{r}$ *satisfies* $P \leq S$. *Then* $(S_N)_{x,0} = \infty$ *for all* $x \in \mathcal{C}$.

Proof. ► **3** $\Rightarrow$ **1**. Let S' be such that for all SMs Q, $Q \leq S'$ if, and only if there is $Q' \leq S_N$ with $(N', Q') = \mathsf{Pre}_{\mathsf{time}}((s, Q))$ and $\mathsf{N}' = \mathsf{N}$. We will show that S' satisfies the required condition. We first assume that $Q \leq S'$. By Lemma 7.2.5, there exists $Q'' \leq S_N$ such that $(N'', Q'') = \mathsf{shrink}^+((N', Q')) = \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}((s, Q)))$ for $\mathsf{N}'' = \mathsf{N}$. And by definition of S_N, there exists $P \leq S$ such that $(r', P) = r \cap (N'', Q'')$ for some $\mathsf{r}' = \mathsf{r}$.

Conversely, if $Q \not\leq S'$, then if Q' denotes the SM such that $(N', Q') = \mathsf{Pre}_{\mathsf{time}}((s, Q))$, then $Q' \not\leq S_N$. By Lemma 7.2.5, if Q'' denotes the SM such that $(N'', Q'') = \mathsf{shrink}^+((N', Q'))$, then $Q'' \not\leq S_N$. Therefore, there is no SM P such that $(r', P) = r \cap \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}((s, Q)))$.

► **not 2** $\Rightarrow$ **not 1**. Let $\langle V, S'\rangle = \mathsf{neighbor}\langle r, S\rangle$. We assume that $V \not\subseteq N$. Then, if t denotes the region $s \lessdot t$, we have $s, t \subseteq V$. By definition of the neighborhood, for any SM $P \leq S_r$, when δ is small enough, there exists a valuation $v \in r - \delta P$ such that $v + d' \in t$ for some $0 \leq d' \leq \delta$. But since $s \lessdot t$, $v + d' \in s$ for some $d' \geq 0$ implies that $0 \leq d' < \delta$ (and there exists $0 \leq d' < \delta$ with $v + d' \in r'$). Therefore, for any SM Q, if P denotes the SM such that $(r', P) = r \cap \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}((s, Q)))$ with $\mathsf{r}' = \mathsf{r}$, then $P \not\leq S$: indeed, if $P \leq S$, then for small enough $\delta > 0$, $v + [0, \delta] \subseteq \mathsf{Pre}_{\mathsf{time}}(s - \delta Q)$ and in particular $v + \delta \in \mathsf{Pre}_{\mathsf{time}}(s - \delta Q)$, which implies that there is $d'' \geq 0$ such that $v + \delta + d'' \in s - \delta Q \subseteq s$, contradicting the above remark.

► **2** $\Rightarrow$ **3**. Let $(V, S') = \mathsf{neighbor}\langle r, S\rangle$. We have $r \subseteq V \subseteq N$. Let (N', Q) be a normalized shrunk DBM with $\mathsf{N} = \mathsf{N}'$, obtained by setting $Q_{x,0} = 1$ for all $x \in \mathcal{C}$ and 0 all other components. Let Q' be the SM such that $(V', Q') = V \cap (N', Q)$, for some $\mathsf{V}' = \mathsf{V}$.

- We show that $Q' \leq S'$. Define $(Q_1)_{x,y} = Q_{x,y}$ if $\mathsf{V}_{x,y} = \mathsf{N}_{x,y}$ and $(Q_1)_{x,y} = 0$ otherwise. Q' is the normalization of Q_1. We have $S'_{x,0} = S'_{0,x} = \infty$ by Lemma 7.3.1, so $Q'_{x,0} \leq S'_{x,0}$ and $Q'_{0,x} \leq S'_{0,x}$. For any $x, y \in \mathcal{C}$, we assume $S'_{x,y} = 0$. It implies $(S)_{x,y} = 0$. To get a contradiction, assume that there is a path in $\Pi_{x,y}(\mathsf{G}(V))$ with an edge (z, z') such that $\mathsf{V}_{z,z'} = \mathsf{N}_{z,z'}$ and $Q_{z,z'} \geq 1$. Since $Q_{z,z'} \geq 1$, there must be a path in $\Pi_{z,z'}(\mathsf{G}(N))$ that contains an edge $(\alpha, 0)$. But since $\Pi_{z,z'}(\mathsf{G}(N)) \subseteq \Pi_{z,z'}(\mathsf{G}(V))$, there is a path in $\Pi_{x,y}(\mathsf{G}(V))$ that contains the edge $(\alpha, 0)$. This contradicts $S'_{x,y} = 0$ since we know, by Lemma 7.3.1 that $S'_{\alpha,0} = \infty$.

- We now show that $Q \leq S_N$, which implies the desired result. By Lemma 7.3.1, there exists $P \leq S$ such that $(r', P) = r \cap \mathsf{shrink}_{[-\delta,\delta]}((V', Q'))$, for some $\mathsf{r}' = \mathsf{r}$. But, if P' denotes the SM such that $(r'', P') = r \cap (V', Q')$, then $P' \leq P \leq S$ because shrink can only increase the components of the SM. Therefore, $(r'', P') = r \cap V \cap (N, Q) = r \cap (N, Q)$. Thus, by Lemma 7.2.4, we must have $Q \leq S_N$, which implies $(S_N)_{x,0} = \infty$ for all $x \in \mathcal{C}$.

We now assume that the above conditions hold, and prove that S' (of item 1) has the same diagonal constraints as S. By Lemma 7.2.4, S_N is computed as the normalization of S', which is defined for every $x, y \in \mathcal{C}_0$ as $(S')_{x,y} = (S)_{x,y}$ if $\mathsf{N}_{x,y} = \mathsf{r}_{x,y}$ and ∞ otherwise. Since N and r have the same diagonal constraints, $(S)_{x,y} = 0$ for $x, y \in \mathcal{C}$ implies $(S_N)_{x,y} = 0$. If $(S)_{x,y} = \infty$, suppose that $(S_N)_{x,y} = 0$. There exists z, z' such that (x, y) belongs to a path in $\Pi_{z,z'}(\mathsf{G}(N))$, and $(S')_{z,z'} = 0$. But this implies that $\mathsf{r}_{z,z'} = \mathsf{N}_{z,z'}$ and $(S)_{z,z'} = 0$. We have then $\Pi_{z,z'}(\mathsf{G}(N)) \subseteq \Pi_{z,z'}(\mathsf{G}(r))$ since $r \subseteq N$, and this contradicts that $(S)_{x,y} = \infty$. We now show that S' has the same diagonal components as S_N. In fact, S' is the normalization of S'_N defined by $(S'_N)_{0,x} = \infty$ for all $x \in \mathcal{C}$ and $(S'_N)_{x,y} = (S_N)_{x,y}$ for

other $x, y \in \mathcal{C}_0$. Clearly, $(S_N)_{x,y} = 0$ for any $x, y \in \mathcal{C}$ implies $(S')_{x,y} = 0$. Assume that $(S')_{x,y} = 0$ with $x, y \in \mathcal{C}$. Then (x, y) belongs to a path $\pi \in \Pi_{z,z'}(\mathsf{G}(s))$ with $(S'_N)_{z,z'} = 0$, so necessarily $z \neq 0$ and $\mathsf{s}_{z,z'} = \mathsf{N}_{z,z'}$. If π does not contain the node 0, then $\pi \in \Pi_{z,z'}(\mathsf{G}(N))$ since all other weights are the same in $\mathsf{G}(s)$ and $\mathsf{G}(N)$. If it contains node 0, then π still must be in $\Pi_{z,z'}(\mathsf{G}(N))$. In fact, $(S'_N)_{z,z'} = 0$ means that $N_{z,z'} < \infty$ and since all weights are the same in $\mathsf{G}(s)$ and $\mathsf{G}(N)$ except for the edges $(0, z)$ which can only decrease in s, π is a shortest path in both graphs. In both cases, we get $(S_N)_{x,y} = 0$.

That S' is a well shrinking constraint for s follows from the hypothesis that S is a well shrinking constraint for r. In fact, if for some $Q \leq S'$, (s, Q) is empty, then the corresponding (r, P) (defined in item 1) is empty, which is a contradiction. □

Note that this lemma may not hold for all s with $r \lessdot s$. Consider the constrained region $\langle r, S\rangle$ on the right of Fig. 7.5, and let s be the first triangle region above r: any valuation arbitrarily close to the thick segments will be in $r - \delta P$ for any $P \leq S$, but it can only reach s by delaying less than δ time units.

The following lemma describes sets of states from where Controller can ensure reaching a given set, through an action.

Lemma 7.4.2. *Let $\langle r, S\rangle$ be a well constrained region, and let $R \subseteq \mathcal{C}$. Let $\mathcal{N} = \{s_1, \ldots, s_m\}$ be the set of neighboring regions of $\langle r, S\rangle$, and define $t_i = s[R \leftarrow 0]$ for each $1 \leq i \leq m$. Then, there exist well shrinking constraints S_{t_i} for all t_i such that for any set of shrunk DBMs $\left((t'_i, Q_{t'_i})\right)_{1\leq i\leq m}$ with $\mathsf{t_i} = \mathsf{t'_i}$, we have that $Q_{t'_i} \leq S_{t_i}$ for all $1 \leq i \leq m$ if, and only if there exists $P \leq S$ such that*

$$(r', P) \subseteq r \cap \mathit{shrink}_{[-\delta,\delta]}\Big(\bigcup_{1\leq i\leq m} (s \cap \mathit{Unreset}_R((t'_i, Q_{t'_i})))\Big).$$

for some $\mathsf{r'} = \mathsf{r}$. Moreover, all the $\langle t_i, S_{t_i}\rangle$ can be computed in polynomial time.

Proof. It is useful to remember in this proof that $\mathsf{t_i} = \mathsf{t'_i}$ implies $t_i \subseteq t'_i$ since t_i is a region.

Let $\langle N, S'\rangle$ be the (well) constrained neighborhood of $\langle r, S\rangle$, given by Lemma 7.3.1. For any region $s \in \mathcal{N}$, let S_s be the well shrinking constraint given by Lemma 7.2.4, such that for all shrunk DBMs (s', Q) with $\mathsf{s'} = \mathsf{s}$ and $s \subseteq s'$, $Q \leq S_s$ if, and only if there exists $P \leq S'$ with $s \cap (N, P) \subseteq (s', Q)$. Here, S_s is in fact a well shrinking constraint since $s \cap (N, P) \neq \emptyset$ for all $P \leq S'$, by the construction of the neighborhood. Then, let us write $t = s[R \leftarrow 0]$, and let S'_s be the well shrinking constraint given by Lemma 7.2.4, such that for any shrunk DBM (t', Q_t) with $\mathsf{t} = \mathsf{t'}$ and $t \subseteq t'$, $Q_t \leq S'_s$ if, and only if there exists $P_s \leq S_s$ with $(s', P_s) = s \cap \mathsf{Unreset}_R((t', Q_t))$ for some $\mathsf{s'} = \mathsf{s}$ with $s \subseteq s'$. We let $S_t = \min_{s\in\mathcal{N}:t=s[R\leftarrow 0]} S'_s$, where the minimum is taken componentwise. Then S_t is still a well shrinking constraint. Now, S_t satisfies the following property: for any shrunk DBM (t', Q_t) with $\mathsf{t'} = \mathsf{t}$ and $t \subseteq t'$, $Q_t \leq S_t$ if, and only if for all regions $s \in \mathcal{N}$ with $t = s[R \leftarrow 0]$, there exists $P_s \leq S_s$ such that $(s', P_s) = s \cap \mathsf{Unreset}_R((t', Q_t))$ for some s' with $\mathsf{s'} = \mathsf{s}$ and $s \subseteq s'$. Combining this and S' defined above, we get that S_t has the property that for any shrunk DBMs (t', Q_t) with $\mathsf{t'} = \mathsf{t}$ and $t \subseteq t'$, $Q_t \leq S_t$ if, and only if for all $s \in \mathcal{N}$ with $t = s[R \leftarrow 0]$, there exists $P_s \leq S'$ such that $s \cap (N, P_s) \subseteq s \cap \mathsf{Unreset}_R((t', Q_t))$.

For any family of shrunk DBMs $\{(t'_i, Q_{t'_i})\}_{1\leq i\leq m}$ with $t_i \subseteq t'_i$, $\mathsf{t_i} = \mathsf{t'_i}$ and $Q_{t'_i} \leq S_{t_i}$, consider the set $\{P_s\}_{s\in\mathcal{N}}$ as defined above. Let $P' = \max_{s\in\mathcal{N}}(P_s)$, where the max is componentwise. We have $P' \leq S'$ since $P_s \leq S'$ for all $s \in \mathcal{N}$. We have $s \cap (N, P') \subseteq s \cap \mathsf{Unreset}_R((t'_i, Q_{t'_i}))$ for all $s \in \mathcal{N}$,

since $P_s \leq P'$. So,

$$(N, P') = \bigcup_{s \in \mathcal{N}} s \cap (N, P') \subseteq \bigcup_{s \in \mathcal{N}} s \cap \mathsf{Unreset}_R((t'_i, Q_{t'_i})).$$

By Lemma 7.3.1, there exists $P \leq S$ such that

$$(r', P) = r \cap \mathsf{shrink}_{[-\delta,\delta]}((N, P')) \subseteq \bigcup_{s \in \mathcal{N}} s \cap \mathsf{Unreset}_R((t'_i, Q_{t'_i})).$$

This proves the first direction of the lemma.

Consider any SMs $\{Q_{t_i}\}_{1 \leq i \leq m}$ such that $(Q_{t_i})_{x,y} \geq 1$ and $(S_t)_{x,y} = 0$ for some i and $x, y \in \mathcal{C}_0$. Then there exists $r_0 \in \mathcal{N}$ with $r_0[R \leftarrow 0] = t$, such that if Q_{r_0} denotes the SM that satisfies $(r'_0, Q_{r_0}) = r_0 \cap \mathsf{Unreset}_R((t'_i, Q_{t_i}))$ with $\mathsf{r}'_0 = \mathsf{r}_0$ and $r_0 \subseteq r'_0$, we have $Q_{r_0} \not\leq S_{r_0}$. So, there is no SM P_{r_0} such that $r_0 \cap (N, P_{r_0}) \subseteq (r'_0, Q_{r_0})$ and $P_{r_0} \leq S'$. It follows that there is no $P' \leq S'$ satisfying $(N, P') \subseteq \bigcup_{s \in \mathcal{N}} (s', Q_s)$ where $(s', Q_s) = s \cap \mathsf{Unreset}_R((t, Q_t))$. In fact, if this would imply that $(N, P') \cap r_0 \subseteq (r'_0, Q_{r_0})$, since the regions in $\mathcal{N}$ are pairwise disjoint. Therefore, if (N, P') satisfies the above inclusion, then $P' \not\leq S'$, so there is no $P \leq S$ such that (r', P) satisfies the statement of the lemma for some $\mathsf{r}' = \mathsf{r}$. □

This lemma gives for instance the shrinking constraints that should be satisfied in r_1, r_2 and r_3, in Fig. 7.1, once shrinking constraint in r'_0 is known. In this case, the constraint in r'_0 is 0 everywhere since it is a punctual region. The neighborhood $\mathcal{N}$ of r'_0 is composed of r'_0 and two extra regions (defined by $(0 < x < 1) \wedge (x = y)$ and $(1 < x < 2) \wedge (x = y)$). If there are shrinkings of regions r_1, r_2, r_3 satisfying the corresponding shrinking constraints (given in the lemma), and from which Controller wins, then one can derive a shrinking of r'_0, satisfying its constraint, and from which Controller wins.

Note that, in the proof of Lemma 7.4.2, the shrinking matrix P is chosen so as to guarantee the stated inclusion, but it is not always *minimal*. In some cases, one can still satisfy the inclusion while chosing some components smaller. Consequently, for a different choice of P, one could also obtain a smaller δ_0. In general, there may be no minimum P: Consider a region r defined by $1 < x < 2$, $1 < y < 2$, and $x < y$. Given two shrinkings $Q_{0,x} = 1, Q_{y,0} = 2$, and $Q'_{0,x} = 2$, $Q'_{y,0} = 1$, the union $(r - \delta Q) \cup (r - \delta Q')$ is not convex. Then both Q and Q' are minimal choices for P such that $(r, P) \subseteq (r, Q) \cup (r, Q')$.

In the next section, we define the game $\mathcal{RG}(\mathcal{A})$ following this idea, and explain how it captures the game semantics for robustness.

7.5 A Finite Game Abstraction

Let $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E)$ be a timed automaton. We define a finite turn-based game $\mathcal{RG}(\mathcal{A})$ on a graph whose nodes are of two sorts: *square nodes* labelled by (ℓ, r, S_r), where ℓ is a location, r a region, S_r is a well shrinking constraint for r; *diamond nodes* labelled similarly by (ℓ, r, S_r, e) where moreover e is an edge leaving ℓ. Square nodes belong to Controller, while diamond nodes belong to Perturbator. Transitions are defined as follows:

(*a*) From each square node (ℓ, r, S_r), for any edge $e = (\ell, g, \sigma, R, \ell')$ of $\mathcal{A}$, there is a transition to the diamond node (ℓ, s, S_s, e) if the following conditions hold:

(*i*) $r \lessdot^* s$ and $s \subseteq g$;

(*ii*) S_s is such that for all SMs Q, $Q \leq S_s$ iff there exists $P \leq S_r$ with

$$(r', P) = r \cap \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}((s, Q))),$$

for some $\mathsf{r}' = \mathsf{r}$.

(*b*) From each diamond node (ℓ, r, S_r, e), where $e = (\ell, g, \sigma, R, \ell')$ is an edge of $\mathcal{A}$, writing $\mathcal{N}$ for the set of regions in the neighborhood of (r, S_r) and $\mathcal{N}' = \{s[R \leftarrow 0] \mid s \in \mathcal{N}\}$, there are transitions to all square nodes (ℓ', t, S_t) with $t \in \mathcal{N}'$, and $(S_t)_{t \in \mathcal{N}'}$ are such that for all shrunk DBMs $((t', Q_t))_{t \in \mathcal{N}'}$ with $\mathsf{t}' = \mathsf{t}$, it holds $Q_t \leq S_t$ for every $t \in \mathcal{N}'$ iff there exists $P \leq S_r$ such that

$$(r', P) \subseteq r \cap \mathsf{shrink}_{[-\delta,\delta]}(\bigcup_{s \in \mathcal{N}} (s \cap \mathsf{Unreset}_R((t', Q_t))), \qquad \text{where } t = s[R \leftarrow 0],$$

for some $\mathsf{r}' = \mathsf{r}$.

Intuitively, the transitions from the square nodes are the decisions of Controller. In fact, it has to select a delay and a transition whose guard is satisfied. Then Perturbator can choose any region in the neighborhood of the current region, and, after reset, this determines the next state.

Note that $\mathcal{RG}(\mathcal{A})$ can be computed, thanks to Lemmas 7.4.1 and 7.4.2, and has exponential-size. Observe also that $\mathcal{RG}(\mathcal{A})$ is constructed in a forward manner: we start by the initial constrained region (*i.e.* the region of valuation $\mathbf{0}$ with the zero matrix as shrinking constraint), and compute its successors in $\mathcal{RG}(\mathcal{A})$. Then, if Controller has a winning strategy in $\mathcal{RG}(\mathcal{A})$, we construct a winning strategy for $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ by a backward traversal of $\mathcal{RG}(\mathcal{A})$, using Lemmas 7.4.1 and 7.4.2. Thus, we construct $\mathcal{RG}(\mathcal{A})$ by propagating shrinking constraints forward, but later do a backward traversal in it. The correctness of the construction is stated as follows.

Proposition 7.5.1. *For any timed automaton $\mathcal{A}$, Controller has a winning strategy in $\mathcal{RG}(\mathcal{A})$ if, and only if there exists $\delta_0 > 0$ such that Controller wins $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ for all $\delta \in [0, \delta_0)$.*

Note that as we compute a winning strategy for Controller (if any) by Proposition 7.5.1, we can also compute a corresponding δ_0. The upper bound of Theorem 7.1.2 is a consequence of the above proposition, since $\mathcal{RG}(\mathcal{A})$ has exponential size and finite reachability games can be solved in time polynomial in the size of the game.

Changing the perturbation parameters. Consider the semantics where Controller's delays are bounded below by $k\delta$, and the perturbations belong to $[l\delta, m\delta]$ with $l\delta + k\delta \geq 0$, for some rational number $\delta > 0$ and integers k, l, m (Observe that any choice of these rational parameters can be written in this manner). The abstract game construction can then be adapted to this case. In fact, it suffices to replace the operator shrink^+ by $\mathsf{shrink}_{[0,k\delta]}$, and the operator shrink by $\mathsf{shrink}_{[l\delta,m\delta]}$ in the construction.

7.5.1 Proof of Proposition 7.5.1

Controller wins $\mathcal{RG}(\mathcal{A}) \Rightarrow$ Controller wins $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ for small enough $\delta > 0$.

Assume we are given a reachability objective defined by $\ell_\smile \in \mathcal{L}$, and let f be a memoryless winning strategy for Controller in $\mathcal{RG}(\mathcal{A})$ for reaching $\ell_\smile$. Consider the execution tree $\mathcal{T}_f$ of $\mathcal{RG}(\mathcal{A})$, where

Controller plays with f. $\mathcal{T}_f$ is finite by Koenig's Lemma since all branches are winning, thus finite, and all branches end in the target state.

To any square node n of $\mathcal{T}_f$ labelled by (ℓ, r, S_r), we will assign a shrunk DBM $\big(r', P_n\big)$ and $\delta_n > 0$ with $P_n \leq S_r$ and $\mathsf{r} = \mathsf{r}'$, such that Controller wins the game $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ from any state of $\{\ell\} \times (r' - \delta P_n)$ for all $\delta \in [0, \delta_n)$. Remember that $\mathsf{r}' = \mathsf{r}$ implies $r \subseteq r'$ since r is a region. We will define these by a bottom up traversal of $\mathcal{T}_f$. We start by assigning $P_n = \mathbf{0}$ and $\delta_n = \infty$ to all nodes with $\mathsf{loc}(n) = \ell_\smile$ (which are leaves of $\mathcal{T}_f$). These are trivially winning for Controller.

Consider now a square node n of $\mathcal{T}_f$ labelled by (ℓ, r, S_r) whose all square successors have been treated. Then, n has only one successor which is given by f, we assume it is the diamond node n' labelled by (ℓ, s, S_s, e). We write $e = (\ell, g, \sigma, R, \ell')$. Let $\mathcal{N}$ be the set of neighboring regions of $\langle s, S_s \rangle$ given by Lemma 7.3.1. Let $s_1, \ldots, s_m$ be the regions composing $\mathcal{N}$, and let $n'_1, \ldots, n'_m$ denote the successors of n' in $\mathcal{T}_f$, where n'_i is labelled by (ℓ', t_i, S_{t_i}), and such that $t_i = s_i[R \leftarrow 0]$. By construction, shrinking constraints S_{t_i} are given by Lemma 7.4.2 applied to $\langle s, S_s \rangle$. By induction, for each $1 \leq i \leq m$ there is a shrunk DBM $(t'_i, P_{n'_i})$ and $\delta_{n'_i} > 0$ with $\mathsf{t}'_\mathsf{i} = \mathsf{t}_\mathsf{i}$ and $P_{n'_i} \leq S_{t_i}$, such that Controller wins the game $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ from $\{\ell'\} \times (t'_i - \delta P_{n'_i})$ for all $\delta \leq \delta_{n'_i}$. Now, by Lemma 7.4.2, there exists a shrunk DBM (s', Q) with $\mathsf{s}' = \mathsf{s}$ and $Q \leq S_s$ such that

$$s' - \delta Q \subseteq s \cap \mathsf{shrink}_{[-\delta,\delta]}\Big(\bigcup_{1 \leq i \leq m} s_i \cap \mathsf{Unreset}_R(t'_i - \delta P_{n'_i})\Big),$$

for all $0 \leq \delta \leq \delta_{n'}$, where $\delta_{n'} \leq \min_i(\delta_{t_i})$ is computed by Lemma 3.4.7. Then, by construction of the game, there exists $P_n \leq S_r$ and $0 < \delta_n \leq \delta_{n'}$ such that $r' - \delta P_n = r \cap \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}\,(s' - \delta Q))$ for all $0 \leq \delta \leq \delta_n$, for some $\mathsf{r}' = \mathsf{r}$ (Remember that Lemma 7.4.1 applies here on s' since $\mathsf{s} = \mathsf{s}'$ and $s \subseteq s'$). Here, δ_n can be computed using Lemma 3.4.5. Controller wins from these states: in fact, it can delay into $s' - \delta Q$, where, after Perturbator's any move, and the clock resets, the next state belongs to one of the states $t'_i - \delta P_{n'_i}$, which are all winning by induction.

Notice that at each step of the computation, we get non-empty shrunk DBMs satisfying the corresponding shrinking constraints. By construction of $\mathcal{RG}(\mathcal{A})$, we have only well shrinking constraints in all nodes, so the computed shrunk DBMs (r', P_n) are always non-empty. The procedure ends in the initial state $(\ell_0, \mathbf{0}, \mathbf{0})$, so Controller wins $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ by projecting the play to $\mathcal{RG}(\mathcal{A})$ and always staying inside set $r' - \delta P_n$ at any node n, which is possible by construction.

Upper bound on δ_0.

We now assume that Controller has a winning strategy and give an upper bound on δ_0 computed by the algorithm. Consider the set $\mathcal{M}$ of all shrunk DBMs that appear when we construct the winning strategy in the proof above, including the shrunk DBMs corresponding to intermediary results. For instance, in the computation above, given the edge $(\ell_i, r_i, S_{r_i}) \to (\ell_i, r'_i, S_{r'_i}, e)$ and $P_{r'_i}$, we compute P_{r_i} such that

$$(r_i, P_{r_i}) = r \cap \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}\,\big((r'_i, P_{r'_i})\big)).$$

Then, $\mathcal{M}$ contains the shrunk DBMs (r_i, P_{r_i}) and $(r'_i, P_{r'_i})$, but also (M_1, Q_1) such that $(M_1, Q_1) = \mathsf{Pre}_{\mathsf{time}}\,((r_i, P_{r_i}))$, and (M_2, Q_2) such that $(M_2, Q_2) = \mathsf{shrink}^+((M_1, Q_1))$. Now, δ_0 is chosen by the algorithm small enough so that all shrunk DBMs of $\mathcal{M}$ are non-empty and normalized, and all such equations that appear in the construction of a winning strategy above hold, for all $\delta \in [0, \delta_0)$. Using Proposition 3.4.6, one can compute the greatest $\delta_0 > 0$ for which the symbolic strategies encoded by (r_i, P_{r_i}) hold. An upper bound can be also chosen by Remark 3.4.8. In this case, the upper bound could be doubly exponential since the size of the equation is exponential.

Controller loses $\mathcal{RG}(\mathcal{A})$ $\Rightarrow$ Controller loses $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ for any $\delta > 0$.

To prove the theorem, we will assume the Controller loses in $\mathcal{RG}(\mathcal{A})$, and we will construct a winning strategy for the δ-Perturbator in $\mathcal{G}_\delta(\mathcal{A})$ by looking at the projection of the plays in $\mathcal{RG}(\mathcal{A})$, and by imitating the moves of a winning strategy for the Perturbator in $\mathcal{RG}(\mathcal{A})$. The core idea is to show that the Perturbator can always force the game to be close, by some chosen ϵ, to all boundaries of the current region for which the shrinking constraint is 0. This will ensure that from any state of the play, for any move of Controller, Perturbator can force the game to some state inside any successor in $\mathcal{RG}(\mathcal{A})$.

Let us first formalize what we mean by being close to boundaries.

Definition 7.5.2. *Let $\langle M, S\rangle$ any constrained DBM. For any $\epsilon \geq 0$, we say that a valuation ν is* ϵ-tight in M w.r.t. S *if $\nu \in M$, and,*

$$\forall x, y \in \mathcal{C}_0, \quad S_{x,y} = 0 \quad \Rightarrow \quad M_{x,y} - \epsilon \leq \nu(x) - \nu(y). \tag{7.1}$$

We say that M admits tight valuations w.r.t. S *if it admits ϵ-tight valuations for any $\epsilon > 0$.*

We say that $\langle M, S\rangle$ admits tight valuations if for any $P \leq S$, and for all small enough $\delta > 0$, $M - \delta P$ admits tight valuations w.r.t. S.

In the proofs, we will also say that ν is *ϵ-tight in M for a component $(x, y) \in \mathcal{C}_0^2$* when $M_{x,y} - \epsilon \leq \nu(x) - \nu(y)$.

We show that for any move of Controller in $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ from an ϵ-tight valuation, the corresponding edge exists in $\mathcal{RG}(\mathcal{A})$:

Lemma 7.5.3. *Fix $\delta > 0$ and $0 \leq \epsilon \leq \delta/2$. Assume ν is ϵ-tight in r w.r.t. S, and let $\nu' = (\nu + d)$ for some $d \geq \delta$, and some edge $e = (\ell, g, \sigma, R, \ell')$ with $\nu' \models g$. Then, $\mathcal{RG}(\mathcal{A})$ has an edge from (ℓ, r, S_r) to $(\ell, r', S_{r'}, e)$ for $r' = \mathsf{reg}(\nu')$ and for some $S_{r'}$.*

Proof. Let $N = \mathsf{Pre}_{\mathsf{time}}(r')$, and consider, by Lemma 7.2.4, the shrinking constraint S_N such that for any SM Q, $Q \leq S_N$ if, and only if the SM P such that $(r, P) = r \cap (N, Q)$ satisfies $P \leq S_r$. Thanks to Lemma 7.4.1, it is sufficient to show that $(S_N)_{x,0} = \infty$ for all $x \in \mathcal{C}$. To get a contradiction, assume that $(S_N)_{x,0} = 0$ for some $x \in \mathcal{C}$. By (the proof of) Lemma 7.2.4, S_N is the normalization of S'_r defined by $(S'_r)_{x,y} = (S_r)_{x,y}$ if $r_{x,y} = N_{x,y}$ and $(S'_r)_{x,y} = \infty$ otherwise. So, $(S_N)_{x,0} = 0$ means that there exists (z, z') such that $(x, 0)$ is on some path π of $\Pi_{z,z'}(\mathsf{G}(N))$, $r_{z,z'} = N_{z,z'}$ and $(S_r)_{z,z'} = 0$. But since $r \subseteq N$ and $r_{z,z'} = N_{z,z'}$ we have $\Pi_{z,z'}(\mathsf{G}(N)) \subseteq \Pi_{z,z'}(\mathsf{G}(r))$. Moreover, all weights along π are the same in r and N. We get $(S_r)_{x,0} = 0$. On the other hand, since $r' \subseteq N$ and $r \lessdot^* r'$, we have $r_{x,0} \leq r'_{x,0} \leq N_{x,0}$. So $r_{x,0} = r'_{x,0}$. By hypothesis, we have $r_{x,0} - \epsilon \leq \nu(x) \leq r_{x,0}$. But $\epsilon \leq \delta/2$ and $d \geq \delta$, so $\nu(x) + d > r_{x,0} = r'_{x,0}$, a contradiction. □

The following lemma shows that ϵ-tightness is preserved by resets.

Lemma 7.5.4. *Let r, r' be regions such that $r' = r[R \leftarrow 0]$ for some $R \subseteq \mathcal{C}$. Consider shrinking constraints S_r and $S_{r'}$ given by Lemma 7.2.4, such that for any SM Q, $Q \leq S_{r'}$ iff there the SM P such that $(r, P) = r \cap \mathsf{Unreset}_R((r', Q))$ satisfies $P \leq S_r$. Let $\delta > 0$ small enough so that these equations hold for given P and Q. For any $\epsilon > 0$, and any ν that is ϵ-tight in $r - \delta P$ w.r.t. S_r, we have that $\nu' = \nu[R \leftarrow 0]$ is ϵ-tight in $r' - \delta Q$ w.r.t. $S_{r'}$.*

Proof. Let $N = \mathsf{Unreset}_R(r')$ and S_N be the shrinking constraints such that for any SM Q, $Q \leq S_N$ if and only if the SM P such that $r - \delta P = r \cap (N - \delta Q)$ satisfies $P \leq S_r$. Then $S_{r'}$ is such that for

any SM Q, $Q \leq S_{r'}$ if and only if there the SM P such that $N - \delta P = \mathsf{Unreset}_R(r' - \delta Q)$ satisfies $P \leq S_N$. By definition of the reset operation, we have $r'_{x,y} = r_{x,y}$ for all $x, y \notin R$, and $r'_{x,y} = 0$ for all $x, y \in R$ (we assume $0 \in R$). For any $x \in R$ and $y \notin R$, $r'_{x,y} = r_{0,y}$ and $r'_{y,x} = r_{y,0}$.

It suffices to assume that ν is ϵ-tight in $\langle r, S_r\rangle$ and show that $\nu' = \nu[R \leftarrow 0]$ is ϵ-tight in $\langle r', S_{r'}\rangle$. For all $x, y \notin R$ or $x, y \in R$, it is clear that ν' satisfies the ϵ-tightness for these components. Consider $x \in R$ and $y \notin R$ such that $(S_{r'})_{x,y} = 0$. Then $(S_{r'})_{0,y} = 0$, and it suffices to show that $-\nu'(y) \geq r'_{0,y} - \epsilon$ since $\nu'(x) = 0$ and $r_{0,y} = r'_{0,y}$. We are going to show that $(S_r)_{0,y} = 0$. If S'_N denotes the shrinking constraint defined by $(S'_N)_{x,0} = (S'_N)_{0,x} = 0$ for all $x \in R$, $(S'_N)_{x,y} = \infty$ whenever $x \in R$ and $y \in \mathcal{C}$ or inversely, and $(S'_N)_{x,y} = (S_N)_{x,y}$ for $x, y \in \mathcal{C}_0 \setminus R$. Then $S_{r'}$ is the normalization of S'_N. Then, $(S_{r'})_{0,y} = 0$, for $y \notin R$, means that there is $z, z' \in \mathcal{C}_0$ such that for some path $\pi \in \Pi_{z,z'}(\mathsf{G}(r'))$, $(0, y)$ belongs to π and $(S'_N)_{z,z'} = 0$. Then either $z, z' \notin R$ and $(S_N)_{z,z'} = 0$, or $z = 0$ and $z' \in R$.

- Consider the first case. We have $r'_{z,z'} = N_{z,z'}$ (by definition of unreset). We show that there is a path $\pi' \in \Pi_{z,z'}(\mathsf{G}(N))$ that contains $(0, y)$ and whose all nodes are outside R. In fact, $r'_{z,0} = N_{z,0}$ and $r'_{y,z'} = N_{y,z'}$ since $z, z', y \notin R$, and these have finite weights (since π is a shortest path). But N is obtained from r' by setting to ∞ edges with an endpoint in R, and applying normalization. So there must be shortest paths from z to 0, and from y to z' in $\mathsf{G}(N)$. Now, since $(S_N)_{z,z'} = 0$, (z, z') belongs to some path $\pi'' \in \Pi_{\alpha,\beta}(\mathsf{G}(N))$ where $r_{\alpha\beta} = N_{\alpha,\beta}$ and $(S_r)_{\alpha,\beta} = 0$. Moreover, from $r \subseteq N$, $r_{\alpha,\beta} = N_{\alpha,\beta}$ and $r_{z,z'} = N_{z,z'}$, it follows that $\pi', \pi'' \in \Pi_{z,z'}(\mathsf{G}(N)) \subseteq \Pi_{z,z'}(\mathsf{G}(r))$. Then, replacing the edge (z, z') in π'' by the path π', we still get a shortest path in $\mathsf{G}(r)$, that contains $(0, y)$. Therefore, we must have $(S_r)_{0,y} = 0$.

- Assume now that $z = 0$ and $z' \in R$. Assume that π does not contain nodes in R other than z' (if it does, we can shorten π and change z'). Let us write $\pi = z_1 z_2 \dots z_m$ where $z_1 = z$ and $z_m = z'$. Since $r'_{0,z_m} = r'_{z_m,0} = 0$, $\pi' = z_1 \dots z_{m-1}$ is a cycle with weight 0. But since all nodes in π' are outside R, this is also a path in $G(r)$. Therefore $(S_r)_{z_i,z_{i+1}} = 0$ along all edges, and in particular $(S_r)_{0,y} = 0$.

By hypothesis, we have $-\nu(y) \geq r_{0,y} - \epsilon$, and since $r'_{0,y} = r_{0,y}$, we get $-\nu'(y) \geq r'_{0,y} - \epsilon$. The proof of the symmetric case $x \notin R$ and $y \in R$ is similar. □

The next lemma is the main lemma of the proof of the second direction of the theorem. It shows that, starting from an ϵ-tight valuation of $\langle r, S\rangle$, and given a Controller's move, Perturbator can, not only choose any successor available in $\mathcal{RG}(\mathcal{A})$ but also make sure that the resulting valuation is $(\epsilon + \epsilon')$-tight, for arbitrary $\epsilon' > 0$.

Lemma 7.5.5. *Consider a valuation ν that is ϵ-tight in r w.r.t. S. Let $\nu' = (\nu + d)$ for some $d \geq \delta$, let $e = (\ell, g, R, \ell')$ an edge of $\mathcal{A}$ with $\nu' \models g$. Consider the edge of $\mathcal{RG}(\mathcal{A})$ from (ℓ, r, S_r) to $(\ell, r', S_{r'}, e)$ for $r' = \mathsf{reg}(\nu')$ and for some $S_{r'}$, from Lemma 7.5.3. Then, for any region s in the $\langle N, S_N\rangle = \mathsf{neighbor}\langle r', S_{r'}\rangle$, and any $\epsilon' > 0$, there exists $d' \in [d - \delta, d + \delta]$ such that $\nu + d'$ is $(\epsilon + \epsilon')$-tight in s w.r.t. S_s where S_s is such that for all SMs Q, $Q \leq S_s$ if, and only if there is $P \leq S_N$ with $(N, P) \cap s \subseteq (s, Q)$.*

Before proving Lemma 7.5.5, let us first prove the second direction of Proposition 7.5.1.

Proof of Proposition 7.5.1 (Second direction). We fix $\delta > 0$, and consider an arbitrary strategy f for Controller in $\mathcal{G}^{\mathsf{exs}}_\delta(\mathcal{A})$. We are going to define an infinite play π, where Controller follows strategy

f, and the target location is not reached. This proves that Perturbator wins $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ against any strategy of Controller since f is chosen arbitrarily.

By hypothesis, Perturbator has a winning strategy γ in $\mathcal{RG}(\mathcal{A})$. We fix $\epsilon \in [0, \delta/2]$, and we define the sequence $\epsilon_i = \sum_{1 \le j \le i} \frac{\epsilon}{2^j}$ for $i \ge 1$, which is positive and bounded above by ϵ. We construct in parallel a play $\big((\ell_i, r_i, S_{r_i}), (\ell_i, r'_i, S_{r'_i}, e_i)\big)_{i \ge 1}$ of $\mathcal{RG}(\mathcal{A})$ where Perturbator plays with strategy γ. The play $\pi = (\ell_i, \nu_i)_{i \ge 1}$ will satisfy the following invariant:

$$\text{For each state } (\ell_i, \nu_i), \text{ we have } \nu_i \in r_i \text{ and } \nu_i \text{ is } \epsilon_i\text{-tight in } r_i \text{ w.r.t. } S_{r_i}.$$

Initially, we have (ℓ_1, ν_1) with $\nu_1 = \mathbf{0}$, and the initial state $(\ell_1, \mathbf{0}, \mathbf{0})$ of $\mathcal{RG}(\mathcal{A})$ satisfies the invariant. For $i \ge 2$, let us assume that state (ℓ_i, ν_i) of the play satisfies the invariant for the state (ℓ_i, r_i, S_{r_i}) of $\mathcal{RG}(\mathcal{A})$. Let $d \ge \delta$ be the delay and $e_i = (\ell_i, g_i, \sigma_i, R_i, \ell_{i+1})$ the edge prescribed by f from state (ℓ_i, ν_i) given the current history. Let $\nu'_i = \nu_i + d$ and r'_i be the region of ν'_i. Lemma 7.5.3 shows that $\mathcal{RG}(\mathcal{A})$ has the corresponding edge. Let $(\ell_{i+1}, r_{i+1}, S_{r_{i+1}})$ be the successor of $(\ell_i, r'_i, S_{r'_i}, e_i)$ in $\mathcal{RG}(\mathcal{A})$, given by γ. Let s be a region in $N = \mathsf{neighbor}\langle r'_i, S'\rangle$ such that $s[R \leftarrow 0] = r_{i+1}$. From Lemmas 7.5.5 and 7.5.4, it follows that first s, then r_{i+1} are reachable from ν'_i by Perturbator's move, and that the resulting valuation is ϵ_{i+1}-tight in r_{i+1} w.r.t. $S_{r_{i+1}}$ by choosing $\epsilon' = \epsilon/2^{i+1}$ □

Proof of Prop. 7.5.5. Consider such valuations $\nu \in r$ and $\nu' \in r'$. For any $x, y \in \mathcal{C}$, $(S_{r'})_{x,y} = 0$ implies that $r'_{x,y} - \epsilon \le \nu'(x) - \nu'(y) \le r'_{x,y}$, since S_r and $S_{r'}$ have the same diagonal components and time delays do not change the quantities $\nu(x) - \nu(y)$. Hence, ν' is ϵ-tight in r' w.r.t. $S_{r'}$ for all diagonal components. Assume that $r' \lessdot^* s$. The other case is similar. Let S_s be the shrinking constraint such that for all SMs Q, $Q \le S_s$ if and only if there is $P \le S_N$ with $(N, P) \cap s \subseteq (s, Q)$. Let us write the clocks ordered according to their fractional values in r':

$$0 = \mathsf{frac}(X_1) < \mathsf{frac}(X_2) < \ldots < \mathsf{frac}(X_m) < 1,$$

where each X_i is a set of clocks having the same fractional value, and X_1 can be empty.

1. Assume that $(S_{r'})_{x,0} = \infty$ for all $x \in X_2 \cup \ldots \cup X_m$. Then, by definition of the neighborhood, either $r' = s$ or $r' \lessdot s$, where the latter case is possible if $X_1 \neq \emptyset$.

   - If $r' = s$, then Perturbator perturbs by 0. Since $(S_s)_{x,0} = \infty$ and $(S_s)_{x,y} = (S_{r'})_{x,y}$ for all $x, y \in \mathcal{C}$, by Lemma 7.5.6, the valuation ν' is ϵ-tight (s, S_s) for these components. If $(S_s)_{0,x} = 0$ for some $x \in \mathcal{C}$, then $N_{0,x} < r'_{0,x}$ by Lemma 7.5.6 since $s_{0,x} = r_{0,x}$, so we must have $(S_{r'})_{0,x} = 0$, and ν' is also ϵ-tight for components $(0, x)$. Note that the ϵ-tightness of diagonal components follow from the ϵ-tightness of ν in r' w.r.t. $S_{r'}$.
   - If $r' \lessdot s$, we let Perturbator perturb by a positive amount $0 < d' < \epsilon'$, so that the valuation is in s. Since $(S_s)_{x,0} = \infty$ and $(S_s)_{x,y} = (S_{r'})_{x,y}$ for all $x, y \in \mathcal{C}$, by Lemma 7.5.6, for these components, $\nu' + d'$ is ϵ-tight in (s, S_s). If $(S_s)_{0,x} = 0$ for some $x \in \mathcal{C}$, then $N_{0,x} < r'_{0,x}$ by Lemma 7.5.6, so we must have $(S_{r'})_{0,x} = 0$. We have, by hypothesis, $-\nu'(x) \ge r'_{0,x} = s_{0,x} - \epsilon$, so $-(\nu'(x) + d') \ge s_{0,x} - \epsilon - \epsilon'$. $\nu' + d'$ is also ϵ-tight for diagonal components since ν' is.

2. Otherwise, consider the minimum $k \in \{2, \ldots, m\}$ such that $(S_{r'})_{x,0} = 0$ for all $x \in X_k \cup \ldots \cup X_m$. Since s is a successor region of r', there exists $k' \in \{k, \ldots, m\}$ such that either clocks in $X_{k'}$ are integers, or all clocks $X_{k'+1} \cup \ldots \cup X_m$ have changed their integer parts and only these. We treat each case separately:

(a) Assume $s_{x,0} = r'_{x,0}$, $\prec^s_{x,0}{=}\leq$ for $x \in X_{k'}$; $s_{x,0} = r'_{x,0} + 1$ and $\prec^s_{x,0}{=}<$ for all $x \in X_{k'+1} \cup \ldots \cup X_m$; $s_{x,0} = r'_{x,0}$ for all $x \in X_1 \cup \ldots \cup X_{k'-1}$. We define Perturbator's move as $d' = 1 - \mathsf{frac}(\nu'_i(x))$ for $x \in X_{k'}$. It is clear that $\nu' + d' \in s$, and that $d' \leq \epsilon$ since ν' is ϵ-tight. Let us show that the $\nu' + d'$ is ϵ-tight in s w.r.t. S_s. By Lemma 7.5.6, all diagonal constraints in S_s are the same as in $S_{r'}$ so the property is satisfied for these components.

- Let us show ϵ-tightness for components $(x, 0)$ (upper bounds). For all $x \in X_{k'+1} \cup \ldots X_m$, we have $(S_s)_{x,0} = \infty$ by Lemma 7.5.6, since $s_{x,0} = r'_{x,0} + 1 = N_{x,0}$. We have $(\nu' + d')(x) = s_{x,0} = -s_{0,x}$ for $x \in X_{k'}$, so $\nu' + d'$ is ϵ-tight for component $(x, 0)$. For all $x \in X_k \cup \ldots \cup X_{k'-1}$, we have $s_{x,0} = r'_{x,0}$, and $\nu'(x) \geq r'_{x,0} - \epsilon$, which implies $\nu'(x) + d' \geq s_{x,0} - \epsilon$ as required. For all $x \in X_1 \cup \ldots \cup X_{k-1}$, we have $(S_s)_{x,0} = \infty$. In fact, $(S_s)_{x,0} = 0$ means, by Lemma 7.5.6 that $s_{x,0} < N_{x,0}$. But since $r' \preccurlyeq^* s$, we would have $r' \leq s_{x,0} < N_{x,0}$, and this implies that $(S_{r'})_{x,0} = 0$ by definition of N and Lemma 7.5.6. But by the choice of k, and by Proposition 7.3.2, this is a contradiction.

- We now show ϵ-tightness for components $(0, x)$ (lower bounds). For all $x \in X_{k'+1} \cup \ldots \cup X_m$, we have, $-(\nu'(x) + d) \geq s_{0,x} - \epsilon$ since $d' \leq \epsilon$. The property is again clearly satisfied by $\nu'(x) + d'$ for $x \in X_{k'}$ since $\nu'(x) + d' = s_{x,0}$. Consider now $x \in X_1 \cup \ldots \cup X_{k'-1}$ such that $(S_s)_{0,x} = 0$. We have $(S_s)_{0,y} = 0$ for $y \in X_{k'}$, and $(S_s)_{0,x} = 0$, which implies that $(S_s)_{y,x} = 0$ since paths $0, y, x$ and $0, x$ belong to $\mathsf{G}(s)$, and S_s is normalized. Then, we also have $(S_{r'})_{y,x} = 0$, so $\nu'(y) - \nu'(x) \geq s_{y,x} - \epsilon$. means that $\mathsf{frac}(\nu'(x)) + d' \leq \epsilon$. The following figure illustrates this, assuming $x \in X_i$.

$$\underbrace{0 < X_1 < \ldots < X_i}_{\mathsf{frac}(\nu'(x))} \underbrace{< X_{i+1} < \ldots < X_{k'}}_{\geq 1-\epsilon} \underbrace{< \ldots < 1}_{=d'}$$

Therefore $\nu' + d'$ satisfies $\nu'(x) + d' \leq -s_{0,x} + \epsilon$, for all $x \in \mathcal{C}$ such that $(S_s)_{0,x} = 0$.

(b) $s_{x,0} = r'_{x,0} + 1$ and $\prec^s_{x,0}{=}<$ for all $x \in X_{k'} \cup \ldots \cup X_m$; $s_{x,0} = r'_{x,0}$ and $\prec^s_{x,0}{=}<$ for all $x \in X_1 \cup \ldots \cup X_{k'-1}$. In this case, we first delay to the immediate time predecessor of s as in the previous case, then add a positive delay d' of at most ϵ'. Then, $\nu' + d'$ is $(\epsilon + \epsilon')$-tight in s w.r.t. S_s.

□

The following technical lemma is used in the proof of Lemma 7.5.5 above.

Lemma 7.5.6. *Let $\langle N, S_N \rangle$ denote the constrained neighborhood of some constrained region. Let r be any region included in N. Let S_r denote the shrinking constraint such that for all SMs Q, $Q \leq S_r$ if, and only if there is $P \leq S_N$ with $(N, P) \cap r \subseteq (r, Q)$ for all small $\delta > 0$. Then, for all $x, y \in \mathcal{C}$, $(S_r)_{x,y} = 0$ implies that $(S_N)_{x,y} = 0$; for all $x \in \mathcal{C}$, $(S_r)_{x,0} = 0$ implies $r_{x,0} < N_{x,0}$ and $(S_r)_{0,x} = 0$ implies $r_{0,x} < N_{0,x}$.*

Proof. By Lemma 7.2.4, S_r is defined as follows. Let S_1 obtained by $(S_1)_{x,y} = S_{x,y}$ if $r_{x,y} = N_{x,y}$ and 0 otherwise. Then, S_r is the normalization (in the sense of SMs) of S_1, so $S_1 \leq S_r$. But all diagonal constraints are the same in r and N, and $r_{x,0} = N_{x,0}$ implies $(S_1)_{0,x} = S_{x,0} = \infty$, and similarly $r_{0,x} = N_{0,x}$ implies $(S_1)_{0,x} = S_{0,x} = \infty$. The result follows. □

We also note the following corollary of Lemma 7.5.5 and Lemma 7.5.4, which will be used in Section 7.6.

Corollary 7.5.7. *Let (ℓ, r, S_r) be a square node of $\mathcal{RG}(\mathcal{A})$. Then $\langle r, S_r\rangle$ admits tight valuations.*

Proof. We prove this for each node $n = (\ell, r, S_r)$ by induction on the shortest path from the initial node of $\mathcal{RG}(\mathcal{A})$ to n. It is true for the initial node $(\ell_0, \mathbf{0}, \mathbf{0})$. Assume it is the case for $n = (\ell, r, S_r)$, and pick any square node $n'' = (\ell', t, S_t)$ reached via a diamond node $n' = (\ell, s, S_s, e)$. For all successor $(\ell', t', S_{t'})$ of n', pick any SM $Q_{t'} \leq S_{t'}$, and consider $P' \leq S_s$ given by Lemma 7.4.2 such that

$$(s, P') \subseteq s \cap \mathsf{shrink}_{[-\delta,\delta]}(\bigcup_{s' \in \mathcal{N}} (s' \cap \mathsf{Unreset}_R((t', Q_{t'})))),$$

where $\mathcal{N}$ is the neighborhood of $\langle s, S_s\rangle$, and we write $t' = s'[R \leftarrow 0]$ for any $s' \in \mathcal{N}$. Further, let $P_r \leq S_r$ such that

$$(r', P_r) = r \cap \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}((s, P'))).$$

Choose $\delta > 0$ small enough so that all equations hold. Let ν be any ϵ-tight valuation in $r' - \delta P$. By the above equations, by a joint move of Controller and Perturbator, the game can proceed to any of the set $t' - \delta Q_{t'}$. In particular, by Lemmas 7.5.5 and 7.5.4, the resulting valuation can be chosen by Perturbator to be inside $t - \delta Q_t$ as $(\epsilon + \epsilon')$-tight for any $\epsilon' > 0$. Since Q_t was chosen arbitrarily, $\langle t, S_t\rangle$ admits tight valuations.

□

7.6 Extension to Turn-based Timed Games

In this section, we extend the reachability algorithm to turn-based timed games. We consider the excess perturbation game semantics, where Perturbator can suggest *any* delay, including 0.

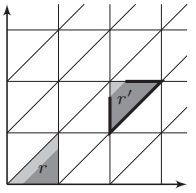
The correctness of the abstract game for timed automata (Proposition 7.5.1) was based on 1) the ability of the controller to always delay inside shrinkings of regions it visits, 2) while not being able to avoid visiting tight valuations. While the former ensured Controller to win $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ if a winning strategy of the abstract is given, the latter ensured the symmetric property for Perturbator.

The situation is different in a game setting since from locations $\mathcal{L}_P$, Perturbator has no reason to delay into particular shrinkings. In fact, consider Fig. 7.6 which shows the states that can be reached by a delay inside region r', at a location $\mathcal{L}_P$, starting from a shrinking of r, with $r \lessdot^+ r'$. Perturbator can thus delay to valuations close to borders. In fact, if the play arrives to $\mathcal{L}_P$ inside $r - \delta P$, then $r' \cap \mathsf{Post}_{\mathsf{time}}(r - \delta P)$ is the set of valuations that that can be reached inside r' after a delay. While this can still be expressed as a shrunk region (by Lemma 3.4.5), this set does not admit tight valuations. We will show however that this set is always included in the union of at most two shrunk regions that admit tight valuations (See Fig. 7.6(b)). This property will allow us to adapt the abstract game construction and prove its correctness following the same proof as for timed automata.

For the proofs, we first need the following characterization of constrained regions that admit tight valuations.

Lemma 7.6.1. *Let $\langle r, S_r\rangle$ be a constrained region. Then, $\langle r, S_r\rangle$ admits tight valuations if, and only if the two following conditions hold:*

1. *For any $x, y \in \mathcal{C}_0$, if $r_{x,y} \neq -r_{y,x}$ and $(S_r)_{x,y} = 0$, then $(S_r)_{y,x} = \infty$,*
2. *For any $x, y \in \mathcal{C}_0$, if there is $x_1 x_2 \dots x_n \in \Pi_{x,y}(\mathcal{G}(r))$ with $(S_r)_{x_i,x_{i+1}} = 0$ for $1 \leq i \leq n-1$, then $(S_r)_{x,y} = 0$.*



(a) A region r and its time successors inside r'.

(b) The shrinking of r' on the left can be decomposed as two shrinkings that admit tight valuations. Any actual delay of Perturbator will belong to one of these two shrinkings.

Figure 7.6: At Perturbator's locations, the delays cannot be expected to end in shrinkings of a particular form. In this example, if Perturbator delays in region r' from r, the valuation can be closed to the borders of the region, and the shrinking expressing the reachable valuations do not admit tight valuations (on the left). We will rather decompose this shrunk region into two shrunk regions that do admit tight valuations (on the right).

Proof. Assume that the first condition is not satisfied. If $r_{x,y} \neq -r_{x,y}$ and $(S_r)_{x,y} = (S_r)_{y,x} = 0$, then for any $\nu \in r$, $\nu(x) - \nu(y) > r_{x,y} - \epsilon$ implies $\nu(x) - \nu(y) > -r_{x,y} + \epsilon$, for small enough $\epsilon > 0$. So no valuation ν is tight in r. Assume now that the second condition is not satisfied. If $(S_r)_{x_i,x_{i+1}} = 0$ for all $1 \leq i \leq n-1$ and $(S_r)_{x_1,x_n} = \infty$, then consider any SM P with $P_{x_1,x_n} = 1$. If ν is ϵ-tight in $r - \delta P$, then we have $r_{x_i,x_{i+1}} - \epsilon \leq \nu(x_i) - \nu(x_{i+1})$ for all $1 \leq i \leq n-1$. Summing these yield $\sum_{i=1}^{n-1} r_{x_i,x_{i+1}} - n\epsilon \leq \nu(x_1) - \nu(x_n)$. We have $\sum_{i=1}^{n-1} r_{x_i,x_{i+1}} = r_{x_1,x_n}$ by hypothesis, and also $\nu(x_1) - \nu(x_n) \leq r - \delta$. Now, this means $r_{x_1,x_n} - \epsilon n \leq r_{x_1,x_n} - \delta$, which is a contradiction for small enough $\epsilon > 0$.

Consider the fractional ordering of the clocks in r: $X_1, \ldots, X_n$ and let $2 \leq i_0 \leq n+1$ and $1 \leq j_0 \leq n$ be the indices given by Lemma 7.3.2. By hypothesis, we must have $j_0 < i_0$, since otherwise $j_0 \geq 2$, and $r_{x,0} \neq -r_{0,x}$ for $x \in X_{j_0}$, but $(S_r)_{x,0} = (S_r)_{0,x} = 0$, contradicting the hypothesis.

Observe that if $x \in X_k$ and $y \in X_l$ with $j_0 < k < l < i_0$, then $(S_r)_{y,x} = \infty$, since $(y, 0, x) \in \Pi_{y,x}(\mathsf{G}(r))$ and $(S_r)_{y,0} = \infty$. Furthermore, unless $i_0 = j_0 + 1$, there exists

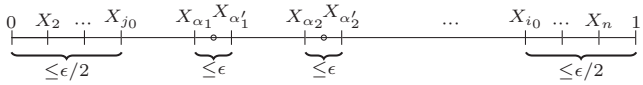
$$j_0 + 1 = \alpha_1 \leq \alpha'_1 < \alpha_2 \leq \alpha'_2 < \ldots < \alpha_m \leq \alpha'_m = i_0 - 1,$$

such that for all $x \in X_k$ and $y \in X_l$ with $\alpha_i \leq k < l \leq \alpha'_i$, we have $(S_r)_{x,y} = 0$, and if $\alpha_i \leq k \leq \alpha'_i$ and $l > \alpha'_i$ then $(S_r)_{x,y} = \infty$. In fact, for any pair $x \in X_k$ and $y \in X_l$ with $j_0 < k < l < i_0$, $(S_r)_{x,y} = 0$ implies that $(S_r)_{x',y'} = 0$ for all $x' \in X_{k'}$ and $y' \in X_{l'}$ with $k \leq k' < l' \leq l$, because S_r is normalized. Moreover, for such a pair x, y, if for $z \in X_{l+1}$, we have $(S_r)_{y,z} = 0$, then also $(S_r)_{x,z} = 0$. This follows from the hypothesis since $x, y, z \in \Pi_{x,z}(\mathsf{G}(r))$ and has S_r-weight equal to 0. Thus, the sequence $(\alpha_i, \alpha'_i)_i$ is well-defined. Consider $x \in X_k$ and $y \in X_l$ for $k \in [\alpha_i, \alpha'_i]$ and $y \in [\alpha_j, \alpha'_j]$ with $i < j$, then $(S_r)_{x,y} = \infty$. In fact, for any clock $z \in X_{\alpha'_i}$ and $z' \in X_{\alpha_j}$, we have $x, z, z', y \in \Pi_{x,y}(\mathsf{G}(r))$, and $(S_r)_{z,z'} = \infty$ by definition of the sequence α, α'.

We now show that (r, S_r) has ϵ-tight valuations. Consider any normalized SM $P \leq S_r$ and $1/\delta > 2(|\mathcal{C}| + 2) \max_{x,y \in \mathcal{C}_0}(P_{x,y})$.

Fix any $0 < \epsilon < \delta/2$. We define a valuation ν as follows. We let $\nu(x) = r_{x,0}$ for $x \in X_1$.

We choose the values $\nu(x)$ for $x \in X_2 \cup \ldots \cup X_{j_0}$ as $-r_{0,x} < \nu(x) < -r_{0,x} + \epsilon/2$, such that they respect the fractional ordering. Similarly, we choose $r_{x,0} - \epsilon/2 < \nu(x) < r_{x,0}$ for $x \in X_{i_0} \cup \ldots \cup X_n$, respecting the fractional ordering. Define $A_i = \frac{i}{|\mathcal{C}|+2}$ for $1 \leq i \leq m$. Define $X_{i,j} = X_i \cup X_{i+1} \cup \ldots X_j$. For each set X_{α_i,α'_i} for $1 \leq i \leq m$, we choose values in the interval $[A_i - \epsilon/2, A_i + \epsilon/2]$ respecting the fractional orderings. The valuation is illustrated in the following figure.



We show that ν is ϵ-tight in $r - \delta P$. We verify at the same time that ν belongs to $r - \delta P$, and that it is ϵ-tight. We first consider the constraints between clocks $X_2 \cup \ldots X_{j_0}$ and $X_{i_0} \cup \ldots \cup X_n$, then between clocks in $x_{j_0+1} \cup \ldots \cup X_{i_0-1}$, and finally for pairs between the two sets.

1. For all $x \in X_2 \cup \ldots \cup X_{j_0}$, we have, by definition $(S_r)_{0,x} = 0$ and $-r_{0,x} < \nu(x) < -r_{0,x} + \epsilon/2$, and $(S_r)_{x,0} = \infty$. By the choice of δ, $\nu(x) < -r_{0,x} + \epsilon/2 < r_{x,0} - \delta Q_{x,0}$. For any pair $x \in X_k$ and $y \in X_l$ with $1 \leq k < l \leq j_0$, we have $(S_r)_{x,y} = 0$ by normalization since $0, x, y$ is in $\Pi_{0,y}(\mathsf{G}(r))$, but the previous inequality implies that $r_{x,y} - \epsilon/2 < \nu(x) - \nu(y) < r_{x,y}$. We have, therefore, $(S_r)_{y,x} = \infty$, and $\nu(y) - \nu(x) < -r_{x,y} + \epsilon/2 < r_{y,x} - \delta P_{x,y}$ by the choice of δ.

2. Similarly, for all $x \in X_{i_0} \cup \ldots \cup X_n$, we have $(S_r)_{0,x} = \infty$, $(S_r)_{x,0} = 0$ and $r_{x,0} - \epsilon/2 < \nu(x) < r_{x,0}$. For any pair $x \in X_k$ and $y \in X_l$ with $i_0 \leq k < l \leq n$, we have $(S_r)_{x,y} = 0$ as before, and $r_{x,y} - \epsilon/2 < \nu(x) - \nu(y) < r_{x,y}$. We have $(S_r)_{y,x} = \infty$, and $\nu(y) - \nu(x) < -r_{x,y} + \epsilon/2 < r_{y,x} - \delta P_{y,x}$ by the choice of δ and ϵ.

3. For any pair $x \in X_2 \cup \ldots \cup X_{j_0}$ and $y \in X_{i_0} \cup \ldots \cup X_n$, we have $(S_r)_{y,x} = 0$ by hypothesis, since $y, 0, x$ is in $\Pi_{y,x}(\mathsf{G}(r))$ and all edges have S_r-weight 0. Thus, $\nu(y) - \nu(x) < r_{y,x} - \delta P_{y,x}$, and we have $r_{y,x} - \epsilon < \nu(y) - \nu(x)$ since $\nu(x) > r_{x,0} - \epsilon/2$ and $\nu(y) < -r_{0,y} + \epsilon/2$. Furthermore, $-r_{x,y} + \delta P_{x,y} < \nu(y) - \nu(x)$ since $-r_{x,y} + \delta P_{x,y} < r_{y,x} - \epsilon$ by the choice of δ and ϵ.

4. Let us now consider constraints between clocks in $X_{j_0+1} \cup \ldots X_{i_0-1}$. For any $x \in X_k$ and $y \in X_l$ with $j_0 < k < l < i_0$, we have $(S_r)_{y,x} = \infty$ since $(y, 0, x)$ belongs to $\Pi_{y,x}(\mathsf{G}(r))$ and $(S_r)_{y,0} = \infty$. We have $\nu(y) - \nu(x) < r_{y,x} - \delta P_{y,x}$ since $\mathsf{frac}(\mathsf{frac}(\nu(y)) - \mathsf{frac}(\nu(x))) \leq \frac{m-1}{|\mathcal{C}|+2} + \epsilon \leq 1 - \delta P_{y,x}$. Assume that $\alpha_i \leq k < l \leq \alpha'_i$, for some i. We have $(S_r)_{x,y} = 0$ and $r_{x,y} - \epsilon < \nu(x) - \nu(y) < r_{x,y}$ by definition of ν. If $k \in [\alpha_i, \alpha'_i]$ and $l \in [\alpha_j, \alpha'_j]$ for $i < j$, then $(S_r)_{x,y} = \infty$. Since $\frac{1}{|C|+2} + \epsilon > \delta P_{x,y}$, we have $\nu(x) - \nu(y) < r_{x,y} - \delta P_{x,y}$.

5. Consider now clock $x \in X_2 \cup \ldots \cup X_{j_0}$ and $y \in X_{j_0+1} \cup \ldots \cup X_{i_0-1}$. We have $(S_r)_{x,y} = \infty$ by hypothesis. In fact, $(0, x, y) \in \Pi_{0,y}(\mathsf{G}(r))$ and $(S_r)_{x,y} = 0$ would imply $(S_r)_{0,y} = 0$. Also, $(S_r)_{y,x} = \infty$ because $(S_r)_{y,0} = \infty$. We have $-r_{y,x} + \delta P_{y,x} < \nu(x) - \nu(y) < r_{x,y} - \delta P_{x,y}$ by $\frac{1}{|C|+2} + \epsilon > \delta \max_{x,y} P_{x,y}$.

□

We extend Lemma 7.2.4 to the $\mathsf{Post}_{\mathsf{time}}()$ operation.

Lemma 7.6.2. *Let M, N be normalized non-empty DBMs such that $N = \mathsf{Post}_{\mathsf{time}}(M)$ and $S \leq \mathsf{Well}(M)$. There exists a shrinking constraint S' for N such that for all SMs Q, $Q \leq S'$ if, and only if there exists $P \leq S$ with $\mathsf{Post}_{\mathsf{time}}((M, P)) = (N', Q)$ with $\mathsf{N}' = \mathsf{N}$.*

Proof. Recall that N is obtained from M by setting all components $(x,0)$ to $(\infty,<)$ and that the resulting DBM is already normalized. Define $(S_1)_{x,0} = \infty$ for all $x \in \mathcal{C}$ and $(S_1)_{x,y} = S_{x,y}$ for all $(x,y) \in \mathcal{C}_0 \times \mathcal{C}$. We define S' as the normalization of S_1.

For any $P \leq S$, let (N,Q) denote the shrunk DBM such that $(N,Q) = \mathsf{Post}_{\mathsf{time}}((M,P))$. Let us show that $Q \leq S'$. Here, Q is the normalization of Q' defined by $Q'_{x,y} = P_{x,y}$ if $y \neq 0$ and $Q'_{x,y} = 0$ if $y = 0$. Since $N_{x,0} = \infty$ for all $x \in \mathcal{C}$, we only need to verify that $Q'_{x,y} \leq S'_{x,y}$ when $y \neq 0$. For these components, we have $N_{x,y} = M_{x,y}$ and $M \subseteq N$, therefore $\Pi_{x,y}(\mathsf{G}(N)) \subseteq \Pi_{x,y}(\mathsf{G}(M))$. So Q' is already normalized because P is, and $Q = Q'$. By definition of S_1 above, we have $Q \leq S_1$, which implies $Q \leq S'$ (since Q is normalized). This also shows that if $Q \not\leq S'$, then for all $P \leq S$, we have $\mathsf{Post}_{\mathsf{time}}(M - \delta P) \not\subseteq N - \delta Q$. In fact, for any such P, as we just showed, there exists $Q' \leq S'$ such that $\mathsf{Post}_{\mathsf{time}}(M - \delta P) = N - \delta Q'$. But $N - \delta Q' \subseteq N - \delta Q$ if, and only if $Q \leq Q'$, and $Q' \leq S'$ implies $Q \leq S$.

For any $Q \leq S'$, let us define P such that $(M,P) = M \cap (N,Q)$. We have $\mathsf{Post}_{\mathsf{time}}((M,P)) = (N,Q)$, since $\mathsf{Post}_{\mathsf{time}}(M \cap (N,Q)) = (N,Q)$. Let us show that $P \leq S$. P is the normalization of P' defined by $P'_{x,y} = Q_{x,y}$ if $y \neq 0$ and $P'_{x,y} = 0$ otherwise. Consider (x,y) such that $S_{x,y} = 0$.

- If $y \neq 0$, then $S'_{x,y} = 0$, by definition of S'. Consider any path $x_1 \dots x_n$ in $\Pi_{x,y}(\mathsf{G}(M))$. We have $S_{x_j,x_{j+1}} = 0$ for all $1 \leq j \leq n-1$. But, if $x_{j+1} = 0$, then $P'_{x_j,x_{j+1}} = 0$ by definition, and otherwise $P'_{x_j,x_{j+1}} = 0$ because $S'_{x_j,x_{j+1}} \leq S_{x_j,x_{j+1}} = 0$. Therefore, $P_{x,y} \leq S_{x,y}$.

- For the case $y = 0$, one can notice that since $N_{x,0} = \infty$ for all $x \in \mathcal{C}$, for all normalized SMs P, we have $P_{x,0} = 0$ by definition. □

The following lemma gives the construction illustrated in Fig. 7.6.

Lemma 7.6.3. *Let $\langle r, S_r\rangle$ be any constrained region that admit tight valuations, and r' such that $r \preccurlyeq^* r'$. There exist, and one can compute, $S^1, \dots, S^m$ with $m \in \{1,2\}$, such that each $\langle r', S^i\rangle$ admits tight valuations and for any SMs $(Q_i)_{1\leq i\leq m}$, we have $Q_i \leq S^i$ for all $1 \leq i \leq m$, if, and only if there exists an SM $P \leq S_r$ with*

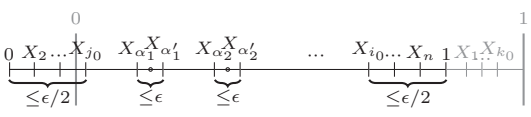
$$\mathsf{Post}_{\mathsf{time}}(r - \delta P) \cap r' \subseteq \bigcup_{i=1}^{m} (r' - \delta Q_i),$$

for all small $\delta > 0$. Moreover, in this case, for any ϵ-tight valuation $v \in (r,P)$ and any $i \in \{1,2\}$ and $\epsilon' > 0$, there exists $d \geq 0$ such that $v + d$ is $(\epsilon + \epsilon')$-tight in (r', Q_i).

Proof. Consider the clock ordering of a constrained region with tight valuations as in the proof of Lemma 7.6.1. We assume the indices $1 \leq i_0, j_0 \leq n$ and α_i, α'_i's are defined (see figure below). We distinguish cases according to the last clock that crosses an integer value during the delay from r to r': There exists $1 \leq k_0 \leq n$ and $m \in \mathbb{N}$ such that for all $x \in X_1 \cup \dots \cup X_{k_0}$, $-r'_{0,x} = -r_{0,x} + m$, while for all $x \in X_{k_0+1} \cup \dots \cup X_n$, we have $-r'_{0,x} = -r_{0,x} + m + 1$. Notice that $-r_{0,x}$ is the integer part of the clock x in region r. If all integer parts have grown by m, then we let $k_0 = 1$. In the following figure, we represent by a vertical gray line some possible values of the index k_0.

We distinguish several cases according to the relative position of k_0.

1. Assume that $1 \leq k_0 \leq j_0 - 1$. The clock ordering before the delay is illustrated in the following figure in black, whereas the ordering after the delay is given between the gray 0 and 1.



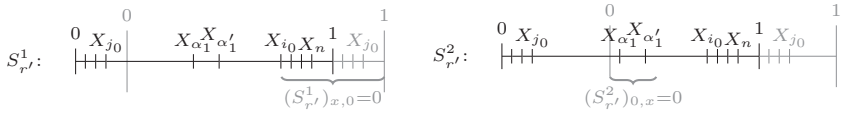
Let us consider the shrinking constraint $S_{r'}$ given by Lemmas 7.2.4 and 7.6.2, so that for all SMs Q, we have $Q \leq S_{r'}$ iff there is $P \leq S$ with $\mathsf{Post}_{\mathsf{time}}(r - \delta P) \cap r' \subseteq r' - \delta Q$. The constraint $S_{r'}$ can be obtained from S_r by only changing the following components: for all $x \in X_1 \cup \ldots \cup X_{k_0}$ $(S_{r'})_{x,0} = 0$, and either $r'_{x,0} = -r'_{x,0}$ and $(S_{r'})_{0,x} = 0$ or $(S_{r'})_{0,x} = \infty$. In fact, since $r' - \delta Q$ contains the successors of $r - \delta P$, the diagonal constraints are the same. Moreover, the partition of the clocks given by i_0, j_0 and α_i, α'_i's are preserved by delays; while time is only translated. So the clocks $X_0, \ldots, X_{k_0}$ will be mixed with $X_{i_0}, \ldots, X_n$ but the constraints on the rest of the clocks will remain. By Lemma 7.6.1, $(r', S_{r'})$ has tight valuations.

Consider SMs Q and P such that $\mathsf{Post}_{\mathsf{time}}(r - \delta P) \cap r' \subseteq r' - \delta Q$ for all small enough $\delta > 0$. Let $\nu \in r - \delta P$ be an ϵ-tight valuation in (r, S_r). Define $d = m + 1 - \nu(x_{k_0+1})$ if x_{k_0+1} has integer value in r' and $d = m + 1 - \frac{\nu(x_{k_0}) + \nu(x_{k_0+1})}{2}$ otherwise (if $k_0 = n$, then let us assume $x_{n+1} = 1$). Then, according to the definition of k_0 and m, $\nu + d$ belongs to r'. Therefore, also $\nu + d \in r - \delta Q$. We show that $\nu + d$ is ϵ-tight in $(r', S_{r'})$. Diagonal components stay unchanged both in shrinking constraints and in valuations, so we only need to verify the constraints $(S_{r'})_{x,0}$ and $(S_{r'})_{0,x}$. Since the distances between the first j_0 clocks are at most ϵ in ν (i.e. $\nu(x_{j_0}) - (-r_{0,x_{j_0}}) \leq \epsilon$), we have that $r'_{x,0} - \epsilon \leq \nu(x) + d \leq r'_{x,0}$ for all $x \in X_1 \cup \ldots \cup X_{k_0}$, and $-r'_{0,x} \leq \nu(x) + d \leq -r'_{0,x} + \epsilon$ for all $x \in X_{k_0+1} \cup \ldots \cup X_{j_0}$. It remains to verify that $r'_{y,0} - \epsilon \leq \nu(y) + d \leq r'_{y,0}$ for $y \in X_{i_0} \cup \ldots \cup X_n$. This follows from the fact that $r_{y,x} - \epsilon \leq \nu(y) - \nu(x) \leq r_{y,x}$ (which holds by hypothesis on (r, S_r)).

The case where $k_0 \geq i_0 + 1$ is symmetric.

2. Assume that $j_0 \leq k_0 < \alpha_1$. Note that we may have $X_{j_0} = \emptyset$ or $X_{i_0} = \emptyset$, or both. We define two shrinking constraints $S^1_{r'}$ and $S^2_{r'}$, such that $(r, S^i_{r'})$ admits tight valuations for both $i = 1, 2$. Both have the same diagonal components as S_r, since these do not change along delays. The first one corresponds to delaying to points where clocks X_{k_0} can be very close to their upper integer values (Fig. 7.7(a)), and the second one to points where clocks X_{α_1} are very close to their integer parts (Fig. 7.7(b)). We will show that for any pair $Q^1 \leq S^1_{r'}$ and $Q^2 \leq S^2_{r'}$, there is a corresponding $P \leq S_r$ such that any valuation obtained by delaying from $r - \delta P$ inside r' lies either in $(r' - \delta Q^1)$ or $(r' - \delta Q^2)$, and moreover ϵ-tight valuations are reached in these shrunk regions if one starts at an ϵ-tight valuation in (r, S_r).

To define $(S^i_{r'})$, we modify S_r as follows. We let $(S^1_{r'})_{x,0} = 0$ and $(S^1_{r'})_{0,x} = \infty$ for all $x \in X_1 \cup \ldots \cup X_{j_0}$. Then $(S^1_{r'})$ admits ϵ-tight valuations by Lemma 7.6.1. We let $(S^2_{r'})_{x,0} = (S^2_{r'})_{0,x} = \infty$ for all $x \in X_1 \cup \ldots \cup X_{j_0}$ and $x \in X_{i_0} \cup \ldots \cup X_n$, and $(S^2_{r'})_{0,x} = 0$ and $(S^2_{r'})_{x,0} = \infty$ for all $x \in X_{\alpha_i} \cup \ldots \cup X_{\alpha'_i}$. Then, similarly, $(S^2_{r'})$ admits ϵ-tight valuations. The definitions are illustrated in Figure 7.7.



(a) $S^1_{r'}$ (b) $S^2_{r'}$

Figure 7.7: The ordering of the clock partition $X_1, \ldots, X_n$ in region r is shown in black in both figures. The new ordering inside r' is shown in gray.

Consider any pair of SMs $Q_i \leq (S^i_{r'})$ for $i \in \{1,2\}$. We define P satisfying S_r as follows. We let $P_{x,0} = 0$ whenever $(S_r)_{x,0} = 0$ or $(S_{r'})_{x,0} = 0$, and $P_{0,x} = 0$ whenever $(S_r)_{0,x} = 0$. We choose arbitrary values for other components satisfying S_r and the following constraints:

(a) $P_{x,y} \geq \max(Q^1_{x,y}, Q^2_{x,y})$ for all $x, y \in \mathcal{C}$. Notice here that $Q^1_{x,y} = Q^2_{x,y} = 0$ whenever $(S_r)_{x,y} = 0$ by definition of $S^1_{r'}$ and $S^2_{r'}$.

(b) For all $x \in X_{\alpha_1} \cup \ldots \cup X_{i_0-1}$, $P_{x,0} \geq \max(Q^1_{x,0}, Q^2_{x,0})$. Notice that $(S_r)_{x,0} = \infty$ for these components.

(c) For all $x \in \mathcal{C} \setminus (X_{\alpha_1} \cup \ldots \cup X_{\alpha'_1})$, $P_{\alpha_1,x} \geq Q^2_{0,x}$. Notice that $(S_r)_{x,\alpha_1} = \infty$ for these components.

(d) For all $x \in \mathcal{C} \setminus (X_{\alpha_1} \cup \ldots \cup X_{\alpha'_1})$, and $y \in X_{\alpha_1} \cup \ldots \cup X_{\alpha'_1}$, $P_{x,\alpha_1} \geq \max(Q^1_{0,x}, Q^2_{x,0} + Q^1_{0,y})$. We have again $(S_r)_{x,\alpha_1} = \infty$.

This shows that one can choose values for P while satisfying S_r. Note also that normalization of P can only increase its components, so all lower bounds given above will still be satisfied.

Consider any $\nu \in r - \delta P$. Let us show that for any $d \geq 0$ with $\nu + d \in r'$, we have $\nu + d \in r' - \delta Q^i$ for $i = 1$ or $i = 2$. For diagonal components $(x, y) \in \mathcal{C}^2$, we have $-r'_{y,x} + \delta(Q_i)_{y,x} \prec^{r'}_{y,x} \nu(x) + d - (\nu(y) + d) \prec^{r'}_{x,y} r'_{x,y} - \delta(Q_i)_{x,y}$ for small enough $\delta > 0$. In fact, either $(S_{r'})_{x,y} = 0$ and $(Q_i)_{x,y} = 0$ for $i = 1, 2$, since $S^i_{r'}$ has the same diagonal components as S_r, or $P_{x,y} \geq \max((Q_1)_{x,y}, (Q_2)_{x,y})$.

It remains to choose $i \in \{1,2\}$ such that the constraints on the components $(x, 0)$ and $(0, x)$ are satisfied. Let us write $\nu' = \nu + d$.

(a) Assume that for all $x \in X_1 \cup \ldots X_n$, $-r'_{0,x} + \delta Q^1_{0,x} \prec^{r'}_{0,x} \nu'(x)$. For all $x \in X_{i_0} \cup \ldots \cup X_n$ and $x \in X_1 \cup \ldots \cup X_{j_0}$, we have $Q^1_{x,0} = 0$ so $\nu(x) \prec^{r'}_{x,0} r'_{x,0} - \delta Q^1_{x,0}$. For any $x \in X_{\alpha_1} \cup \ldots X_{i_0-1}$, we have $P_{x,0} \geq Q^1_{x,0}$, so $\nu'(x) \prec^{r'}_{x,0} r'_{x,0} - \delta Q^1_{x,0}$.

(b) Assume that for some $1 \leq i \leq n$, for clocks $x \in X_i$, $-r'_{0,x} + \delta Q^1_{0,x} \not\prec^{r'}_{0,x} \nu'(x)$. We show that $\nu' \in r' - \delta Q^2$. Let us first show that we have $i \in \{\alpha_1, \ldots, \alpha'_1\}$. This follows from the fact that $P_{x,\alpha_1} \geq Q^1_{0,x}$ for all clocks $x \in \mathcal{C} \setminus (X_{\alpha_1} \cup \ldots \cup X_{\alpha'_1})$. In fact, this ensures that the constraint $-r'_{0,x} + \delta Q^1_{0,x} \prec^{r'}_{0,x} \nu'(x)$ is always satisfied for these clocks, hence $i \in \{\alpha_1, \ldots, \alpha'_1\}$.

We first show that the upper bounds hold. Let $y \in X_{\alpha_1}$ and $y' \in X_i$. We have $\nu'(y) \leq -r'_{0,y} + \delta Q^1_{0,y'}$ since $\nu'(y') \leq -r'_{0,y'} + \delta Q^1_{0,y'}$ by definition of i, and $\mathsf{frac}(\nu(y)) \leq \mathsf{frac}(\nu(y'))$. For any $x \in X_{i_0} \cup \ldots \cup X_n \cup X_1 \cup \ldots \cup X_{j_0}$, we also have $\nu'(x) - \nu'(y) \prec^{r'}_{x,y} r'_{x,y} - \delta P_{x,y}$, which together yields $\nu'(x) \prec^{r'}_{x,0} r'_{x,0} - \delta(P_{x,y} - Q^1_{0,y'})$. This implies the desired bound since $P_{x,y} \geq Q^2_{x,0} + Q^1_{0,y'}$. For any clock $x \in X_{\alpha_1} \cup \ldots \cup X_{i_0-1}$, we have $P_{x,0} \geq Q^2_{x,0}$, which yields $\nu'(x) \prec^{r'}_{x,0} -\delta Q^2_{x,0}$.

It remains to show the lower bounds. We have $Q^2_{0,x} = 0$ for all $x \in X_{\alpha_1} \cup \ldots \cup X_{\alpha'_1}$ so $-r'_{0,x} + \delta Q^2_{0,x} \prec^{r'}_{0,x} \nu'(x)$. For all $x \in \mathcal{C} \setminus (X_{\alpha_1} \cup \ldots \cup X_{\alpha'_1})$, we have by definition $P_{\alpha_1,x} \geq Q^2_{0,x}$. Then, $-r'_{\alpha_1,x} + \delta P_{\alpha_1,x} \prec^{r'}_{\alpha_1,x} \nu'(x) - \nu'(\alpha_1)$. Combining with the fact that $\nu'(\alpha_1) \geq -r'_{0,\alpha_1}$ we get $-r'_{0,x} + \delta Q^2_{0,x} \prec^{r'}_{0,x} \nu'(x)$.

It is now easy to see that from ϵ-tight valuations in (r, S_r), one can reach $(\epsilon + \epsilon')$-tight valuations in both $(r', S^1_{r'})$ and $(r, S^2_{r'})$. In fact, for any $\epsilon' > 0$, the delay $d \geq 0$ can be

chosen such that $r'_{x,0} - \nu'(x) \leq \epsilon'$, for $x \in X_{j_0}$, but also such that $\nu'(x) + r'_{0,x} \leq \epsilon'$ for $x \in X_{\alpha_1}$. In the former case, ν' is $(\epsilon + \epsilon')$-tight in $(r', S^1_{r'})$, and in the latter case, ν' is $(\epsilon + \epsilon')$-tight in $(r', S^2_{r'})$.

3. The cases where $\alpha'_i \leq k_0 < \alpha_{i+1}$ for $i > 1$ or $\alpha'_m \leq k_0 < i_0$ are treated similarly to the previous case: Here α'_i has the role of j_0, and α_{i+1} has the role of α_1.

4. Assume that $\alpha_i \leq k_0 < \alpha'_i$. Then $S_{r'}$ has the same diagonal components as S_r, and for any $x \in X_{\alpha_i} \cup \ldots \cup X_{k_0}$, $(S_{r'})_{x,0} = 0$ and $(S_{r'})_{0,x} = \infty$, while for any $x \in X_{k_0+1} \cup \ldots \cup X_{\alpha'_i}$, $(S_{r'})_{x,0} = \infty$ and $(S_{r'})_{0,x} = 0$. Furthermore, for any other clock $x \in \mathcal{C}$, $(S_{r'})_{x,0} = (S_{r'})_{0,x} = \infty$.

   If we fix SMs P and Q such that $\mathsf{Post}_{\mathsf{time}}(r - \delta P) \cap r' \subseteq r' - \delta Q$, then for any ϵ-tight valuation $\nu \in r - \delta P$, any delay d with $\nu + d \in r'$ is ϵ-tight in $(r', S_{r'})$. In fact, the diagonal components stay unchanged during delays, and the fractional values of clocks $X_{\alpha_i} \cup \ldots \cup X_{\alpha'_i}$ differ by at most ϵ in ν.

5. Assume that $j_0 + 1 = i_0$ (there is no α_i, α'_i). Consider the case $k_0 = j_0$. The proof follows again the same ideas as before, but is simpler. We give the details. We define $(S^1_{r'})_{x,y} = (S^2_{r'})_{x,y} = (S_r)_{x,y}$ for all $x, y \in \mathcal{C}$. We let $(S^1_{r'})_{x,0} = 0$ and $(S_{r'})_{0,x} = \infty$ for all $x \in \mathcal{C}$, and $(S^2_{r'})_{0,x} = \infty$ and $(S_{r'})_{0,x} = 0$ for all $x \in \mathcal{C}$. $S^1_{r'}$ is represented in Figure 7.7(a) (with the difference that there is no α_i's).

   Let $Q^i \leq S^i_{r'}$ for $i = 1, 2$. Let $j'_0 = \min\{j \mid 1 \leq j \leq j_0, X_j \neq \emptyset\}$. Define P, where $P_{x,y} \geq Q^1_{0,x}$ for all $x \in X_{i_0} \cup \ldots \cup X_n$ and $y \in X_{j'_0}$. Let $P_{x,i_0} \geq Q^1_{0,y} + Q^2_{x,0}$ for all $x \in \mathcal{C} \setminus X_{i_0}$ and $y \in \mathcal{C}$. Consider $\nu \in r - \delta P$. Let $\nu' = \nu + d$ such that $\nu' \in r'$. If $-r'_{0,x} + \delta \prec^{r'}_{0,x} \nu'(x)$ for all $x \in \mathcal{C}$, then $\nu' \in r' - \delta Q^1$ since $(Q^1)_{x,0} = 0$ and $P_{x,y} \geq Q^1_{x,y}$ for all $x, y \in \mathcal{C}$.

   Otherwise, we have $\nu'(y') \leq -r'_{0,y'} + \delta Q^1_{0,y'}$ for some $y' \in \mathcal{C}$, so $\nu'(i_0) \leq -r'_{0,i_0} + \delta Q^1_{0,y'}$ since $\mathsf{frac}(\nu'(i_0)) \leq \mathsf{frac}(\nu'(y'))$. This means that $\nu'(i_0) \prec^{r'}_{i_0,0} r'_{i_0,0} - \delta Q^2_{x,0}$ for small enough $\delta > 0$. For all $x \in \mathcal{C} \setminus X_{i_0}$, we have $\nu'(x) - \nu'(i_0) \prec^{r'}_{x,i_0} r'_{x,i_0} - \delta P_{x,i_0}$. This implies, by the above constraint on $\nu'(i_0)$ that $\nu'(x) \prec^{r'}_{x,0} r'_{x,0} - \delta(P_{x,i_0} + Q^1_{0,y'})$. This implies $\nu'(x) \prec^{r'}_{x,0} r'_{x,0} - \delta Q^2_{x,0}$ since $P_{x,i_0} \geq Q^1_{0,y'} + Q^2_{x,0}$. We are done since $Q^2_{0,x} = 0$ for all $x \in \mathcal{C}$.

   It is clear that from ϵ-tight valuations in $r - \delta P$, one can delay to $(\epsilon + \epsilon')$-tight valuations both in $r' - \delta Q^1$ and $r' - \delta Q^2$, for any $\epsilon' > 0$.

□

We can now generalize the abstract game to turn-based timed games, and prove the main theorem for these games.

Let us fix a timed game $\mathcal{A} = (\mathcal{L}_C \cup \mathcal{L}_P, \ell_0, \mathcal{C}, \Sigma, E_C, E_P)$. We define $\mathcal{RG}(\mathcal{A})$ as follows. The states of $\mathcal{RG}(\mathcal{A})$ is again given as a set of square nodes (ℓ, r, S_r), where $\ell \in \mathcal{L}_C$ and $\langle r, S_r \rangle$ is a well constrained region. The diamond nodes are now either of the form (ℓ, r, S_r, e) with $\ell \in \mathcal{L}_C$ and e is an edge leaving ℓ, or (ℓ, r, S_r) with $\ell \in \mathcal{L}_P$. As before, square nodes belong to Controller, and diamond nodes belong to Perturbator. The edges leaving square nodes and the diamond nodes of the form (ℓ, r, S_r, e) are defined as for timed automata. There is an edge from a diamond node (ℓ, r, S_r) to (ℓ', t, S_t), if there is an edge $e = (\ell, g, \sigma, R, \ell')$ in $\mathcal{A}$, and the following conditions hold:

i) There is a region s with $r \preccurlyeq^* s$ and $t = s[R \leftarrow 0]$,

ii) Let $m \in \{1,2\}$ and (s, S^i) for $i = \{1, m\}$ be the constrained region(s) given by Lemma 7.6.3 applied to $\langle r, S_r\rangle$. There exists $i \in \{1,2\}$ such that for all SMs P, $P \leq S_t$ iff there exists an SM $Q \leq S^i$ with $(s', Q) = \mathsf{Unreset}_R((t, P))$, for some $\mathsf{s'} = \mathsf{s}$.

Notice that (t, S_t) can be computed in polynomial time thanks to Lemmas 7.6.3 and 7.2.4. Also, all constrained regions of the square nodes admit tight valuations by Lemma 7.6.3 and Corollary 7.5.7.

Theorem 7.1.2, for turn-based timed games, follows from the following proposition.

Proposition 7.6.4. *For any turn-based timed game $\mathcal{A}$, Controller has a winning strategy in $\mathcal{RG}(\mathcal{A})$ if, and only if there exists $\delta_0 > 0$ such that Controller wins $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ for all $\delta \in [0, \delta_0)$.*

Proof. Assume that Controller wins $\mathcal{RG}(\mathcal{A})$. As in the proof of Proposition 7.5.1, we assign to nodes $n = (\ell, r, S_r)$ in $\mathcal{RG}(\mathcal{A})$, shrunk DBMs (r', P_n) with $\mathsf{r'} = \mathsf{r}$, describing the set of winning states from the corresponding locations in $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$. The construction and the proof are the same for square nodes. If n is a diamond node of the form (ℓ, r, S_r), then fix any s with $r \lessdot^* s$ and an edge $e = (\ell, g, R, \ell')$ from n with $s \subseteq g$. Assume, by induction hypothesis, that we have shrunk DBMs (t, P_{n_1}) and (t, P_{n_2}) describing winning states from the successors $n_1 = (\ell', t, S_t^1)$ and $n_2 = (\ell', t, S_t^2)$ of n through the edge e, where $t = s[R \leftarrow 0]$ (Assume $n_1 = n_2$ if there is only one such successor). Consider (s, S^1) and (s, S^2) given by Lemma 7.6.3 applied to $\langle r, S_r\rangle$ (assume again that $S^1 = S^2$ if there is only one such constrained region). By construction, for each $i \in \{1,2\}$, $\langle t, S_t^i\rangle$ is defined by Lemma 7.2.4, such that for all SMs P_{n_i}, $P_{n_i} \leq S_t^i$ iff there exists SM $Q^i \leq S^i$ with $(s_i, Q^i) = \mathsf{Unreset}_R((t, P_{n_i}))$, for some $\mathsf{s_i} = \mathsf{s}$ and $s \subseteq s_i$. Then, by Lemma 7.6.3, there exists P_e^s such that

$$\mathsf{Post}_{\mathsf{time}}((r_e^s, P_e^s)) \cap s \subseteq \bigcup_{i=1}^{m} (s_i, Q_i),$$

for some $\mathsf{r_e^s} = \mathsf{r}$ with $r \subseteq r_e^s$. Here, (r_e^s, P_e^s) describes a set of winning states from location ℓ assuming Perturbator chooses the edge e and delays anywhere in the region s. We define (r_e, P_e) as the intersection of all (r_e^s, P_e^s) with $r \lessdot^* s$, and define (r', P_n) as the intersection of all (r_e, P_e) for all edges e leaving n. This concludes one direction of the proof.

Now, assume that Perturbator wins $\mathcal{RG}(\mathcal{A})$. The proof of Proposition 7.5.1 is based on the construction of a strategy for $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$, that ensured that the valuation is always ϵ-tight inside any visited constrained region. When one projects the play in $\mathcal{RG}(\mathcal{A})$, this property allows Perturbator to choose valuations in $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ belonging to any successor it would choose in $\mathcal{RG}(\mathcal{A})$ in order to win. Thus, Perturbator's strategy simply consists in following in $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ its winning strategy in $\mathcal{RG}(\mathcal{A})$. The same proof carries over to timed games since Lemma 7.6.3 shows that given a ϵ-tight valuation of $\langle r, S_r\rangle$, there exist delays that lead to $(\epsilon + \epsilon')$-tight valuations inside $\langle s, S^1\rangle$ and $\langle s, S^2\rangle$ for any $\epsilon' > 0$. □

7.7 Hardness Result

Proposition 7.7.1. *The parameterized robust reachability problem is* EXPTIME*-hard on timed automata.*

Proof. We use a reduction from the halting problem in linear-bounded alternating Turing machines over a two-letter alphabet $\Sigma = \{a, b\}$. Let $\mathcal{M}$ be such a Turing machine, and write n for the bound on the tape length. We assume w.l.o.g. that $n \geq 3$ and that instructions are of the form:

- (disjunction) $\delta(q) = q' \vee q''$
- (conjunction) $\delta(q) = q' \wedge q''$
- (instruction) $\delta(q) = (\gamma, \gamma', \mathsf{dir}, q')$ where $\gamma, \gamma' \in \{a, b\}$ and $\mathsf{dir} \in \{\leftarrow, \rightarrow\}$. Such a transition reads a γ in the current cell, write a γ' and follows direction given by dir.

Our encoding of $\mathcal{M}$ uses the set of $n+2$ clocks $X = \{x_i \mid i = 1 \dots n\} \cup \{y, z\}$. The content of cell i is encoded by clock x_i: it is an a if the value of clock x_i is $n-i$, and a b if the value of clock x_i is bounded below by $2n-i$. Due to robustness concerns, these guards will be relaxed a bit in the construction.

We assume the content of the tape is represented by a word w over alphabet Σ of length n. Let $k \in \mathbb{N}$ and $\epsilon \geq 0$. We say that a valuation v over X is a *k-shift encoding of w with precision ϵ* whenever $v(y) = v(z) = 0$, and for every $1 \leq i \leq n$:

- $w_i = a$ iff $-\epsilon \leq v(x_i) - (n-i) - k \leq \epsilon$
- $w_i = b$ iff $-\epsilon \leq v(x_i) - (2n-i) - k$

We encode the instructions as follows.

▶ **Regular instruction.** A transition $\delta(q) = (\gamma, \gamma', \mathsf{dir}, q')$ is mimicked thanks to modules $\mathsf{instr}^{i,k}_{\delta(q)=(\gamma,\gamma',\mathsf{dir},q')}$ for every $1 \leq i \leq n$ and $0 \leq k < n$. Such a module is depicted on Fig. 7.9: it is a sequence of n modules, the ith one being of a special shape. The initial state is $(q, i, k, 1)$. We write I for the interval $[n-1, n+1]$ and I' for the interval $[2n-1, +\infty)$, and adopt the notation $S + k = \{b + k \mid b \in S\}$ for any set S.

The correctness of this module is given by the following lemma:

Lemma 7.7.2. *Let $0 \leq \epsilon \leq 1$ and $0 \leq \delta \leq 1$. Let $w \in \Sigma^n$ such that $w_i = \gamma$. Assume module $\mathsf{instr}^{i,k}_{\delta(q)=(\gamma,\gamma',\mathsf{dir},q')}$ is entered with valuation v which is a k-shift encoding of w with precision ϵ. The Controller has a unique strategy in this module, and for every response of the δ-perturbator, the valuation v' when leaving the module is a 0-shift encoding of $w[w_i \leftarrow \gamma']$ (the word obtained from w by replacing w_i with γ') with precision 2δ.*

Proof. There is a unique strategy for Controller, which is to play from state (q, i, k, j) when z reaches j with the unique possible transition: if initially $|v(x_j) - (n-j) - k| \leq \epsilon$, then he will choose the top-most transition, and if initially $v(x_j) \geq 2n - j + k - \epsilon$, then it will choose the bottom-most transition. Indeed note that in the first case, since $\epsilon \leq 1$, the value of clock x_j when z reaches j lies within $I + k$. When clock x_j is reset clock z is almost j (more precisely it lies between $j - \delta$ and $j + \delta$), whereas clock z is almost n when leaving the module (more precisely it lies between $n - \delta$ and $n + \delta$). In the second case, the value of x_j is increased by almost n. This straightforwardly implies the mentioned property. □

Remark 7.7.3. *Note that if the module above is entered while $w_i \neq \gamma$, then it reaches a deadlock. This could be avoided using extra transitions to a sink state.*

▶ **Conjunction.** $\delta(q) = q' \wedge q''$ is mimicked thanks to modules $\mathsf{conj}^{i,k}_{\delta(q)=q' \wedge q''}$ (for every $1 \leq i \leq n$ and $0 \leq k < n$) on Figure 7.10. As in the previous module, the Controller has no other choice that selecting the next transition when the constraint is satisfied. For the first transition, the δ-Perturbator can choose to do it a bit earlier, or a bit later, and depending on this, the controller will next to choose either $y = 1 \wedge z \leq 2$ or $y = 1 \wedge z > 2$.

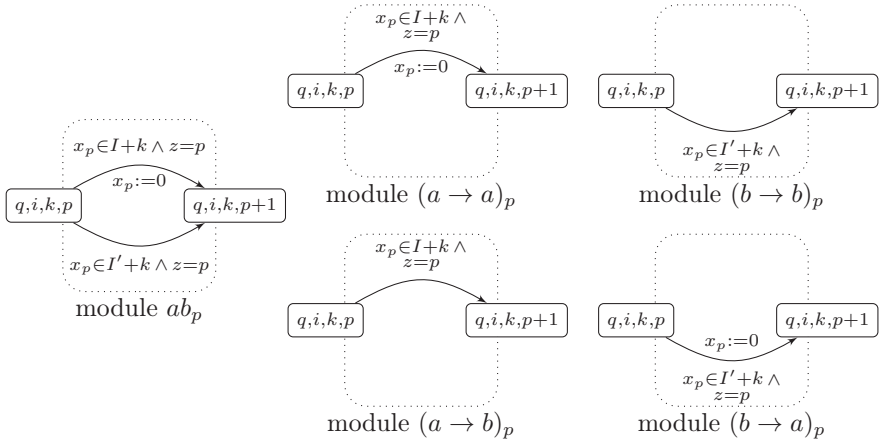


Figure 7.8: Intermediary modules: when traversing module ab_p, either clock x_p belongs to $I + k$ while $z = p$, corresponding to an a at position p. Clock x_p is then reset, so that position p still contains an a; or clock x_p is in $I' + k$ when $z = p$, encoding a b at position p. In that case, clock x_p is not reset, and the content of cell p is preserved. Using similar ideas, modules $(\gamma \to \gamma')_p$, with $\gamma, \gamma' \in \{a, b\}$, check that cell p initially contains γ, and replace it with γ'.

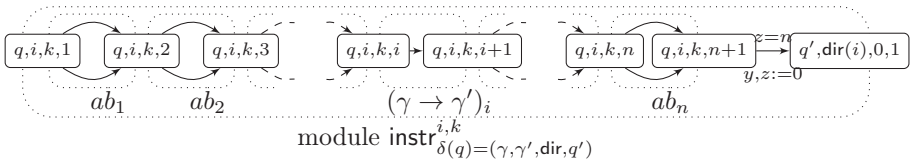


Figure 7.9: Module $\mathsf{instr}^{i,k}_{\delta(q)=(\gamma,\gamma',\mathsf{dir},q')}$ for the simulation of the regular instruction $\delta(q) = (\gamma, \gamma', \mathsf{dir}, q')$, and its constituent submodules. If $\mathsf{dir} = \to$, $\mathsf{dir}(i) = i + 1$ if $1 \leq i < n$, and undefined otherwise. If $\mathsf{dir} = \leftarrow$, $\mathsf{dir}(i) = i - 1$ if $1 < i \leq n$, and undefined otherwise.

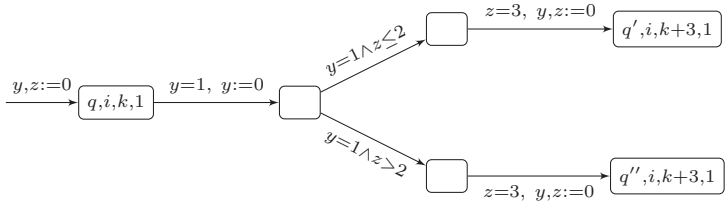


Figure 7.10: Module $\mathsf{conj}^{i,k}_{\delta(q)=q' \wedge q''}$ for conjunctive transition

Lemma 7.7.4. *Let $0 \leq \epsilon \leq 1$ and $0 \leq \delta \leq 1$. Let $w \in \Sigma^n$. Assume module $\mathsf{conj}^{i,k}_{\delta(q)=q' \wedge q''}$ is entered with valuation v which is a k-shift encoding of w with precision ϵ. The Controller has a*

unique strategy in this module, and the δ-Perturbator can choose to reach either $(q', i, k+3, 1)$ or $(q'', i, k+3, 1)$. In both cases, the valuation v' when leaving the module is a $(k+3)$-shift encoding of w with precision $\epsilon + \delta$.

Proof. The Controller has no other choice than satisfying the next constraint of the next transition. On the other hand, the δ-Perturbator can either choose to postpone the first transition, or to fire it earlier. In the first case, the Controller has then to choose the bottom-most transition, and in the second case, the Controller has to choose the top-most transition. Globally the values of the clocks are increased by 3 (plus or minus δ), which yields the expected property. □

► **Disjunction.** $\delta(q) = q' \vee q''$ is mimicked thanks to modules $\mathsf{disj}^{i,k}_{\delta(q)=q'\vee q''}$ (for every $1 \leq i \leq n$ and $0 \leq k < n$) on Figure 7.11.

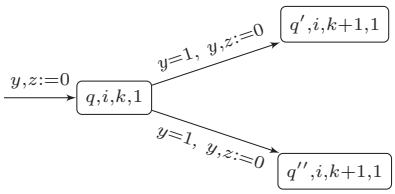


Figure 7.11: Module $\mathsf{disj}^{i,k}_{\delta(q)=q'\wedge q''}$ for disjunctive transition

Lemma 7.7.5. *Let $\epsilon \geq 0$ and $\delta \geq 0$. Let $w \in \Sigma^n$. Assume module $\mathsf{disj}^{i,k}_{\delta(q)=q'\vee q''}$ is entered with valuation v which is a k-shift encoding of w with precision ϵ. The Controller can choose to reach either $(q', i, k+1, 1)$ or $(q'', i, k+1, 1)$. In both cases, the valuation v' when leaving the module is a $(k+1)$-shift encoding of w with precision $\epsilon + \delta$.*

Proof. Similar to the previous proof. □

► **Reset module.** We fix an integer $0 \leq k < n$. Shifts encodings accumulate when stacking disjunctive and conjunctive instructions. We present a module $\mathsf{reset}^{i,k}_q$ which resets the shift from state q, position i.

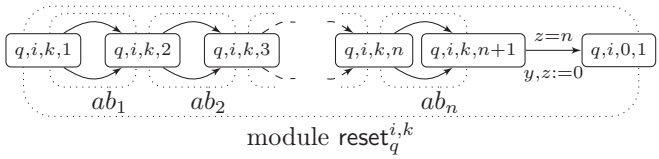


Figure 7.12: Module $\mathsf{reset}^{i,k}_q$ which resets the shift in the encoding.

Lemma 7.7.6. *Let $0 \leq \epsilon \leq 1$ and $0 \leq \delta \leq 1$. Let $w \in \Sigma^n$. Assume module $\mathsf{reset}_{q,i,k}$ is entered with valuation v which is a k-shift encoding of w with precision ϵ. The Controller has a unique strategy in this module, and for every response of the δ-Perturbator, the valuation v' when leaving the module is a 0-shift encoding of w with precision 2δ.*

Proof. This proof is similar to the proof of Lemma 7.7.2. □

▶ **Global reduction.** It remains to glue all the modules together. An easy solution is to apply the reset module after each conjunctive or disjunctive instruction (though there are some more thrifty solutions). The reset module allows both to reset the shift and to reinitialize the imprecision. We write $\mathcal{A}$ for the resulting timed automaton. The halting location of the Turing machine is called final in $\mathcal{A}$.

One can easily check that in $\mathcal{A}$, letting $0 \leq \delta < \frac{1}{2}$, the Controller has a winning strategy against the δ-Perturbator to reach location final if, and only if, the Turing machine $\mathcal{M}$ halts. □

7.8 Conclusion

We considered a game-based approach to robust reachability in timed automata, by modelling the semantics as a game between a controller and its environment. We proved that a bound on the imprecisions, and a corresponding robust strategy for reachability objectives in turn-based timed games can be synthesized, and that the existence of such a bound and a strategy is EXPTIME-complete. The problem is thus harder than classical reachability [AD94]. We believe that the high complexity is due to the ability of Controller to detect whether any perturbation has been observed in a given transition, as in Fig. 7.10, which allows one to use *alternation* when encoding (bounded) Turing machines. In the conservative perturbation game semantics studied in Chapter 8, Controller will not be able to detect perturbations – at least, not immediately. The parameterized robust reachability, and in fact, Büchi objectives then become PSPACE-complete.

A natural continuation of this work would be to look at zone-based on-the-fly algorithms, as for usual timed games in [CDF+05]. In fact, solving usual timed games is also an EXPTIME-complete problem and on-the-fly algorithms allowed efficient implementations.

Developing algorithms for safety objectives, that is, ensuring infinite executions that avoid some given state would require the use of different techniques and seems to be a challenging problem. In fact, in that setting, one has to deal with the accumulation of the imprecisions over infinite runs, which cannot be avoided by adjusting δ.

Chapter 8

Büchi Objectives in Conservative Semantics

8.1 Introduction

The previous chapter established the EXPTIME-completeness of the parameterized robust reachability problem on timed automata under the excess perturbation game semantics. The hardness proof relies on the ability of Controller to detect perturbations after a transition (this allows the encoding of *alternating* linearly bounded Turing machines). A natural question is whether this complexity blow-up, from PSPACE for the exact semantics to EXPTIME, is due to the ability of Controller to detect perturbations. The conservative perturbation game semantics disallows such an ability, since guards with equalities are never enabled. Under this semantics, parameterized robust reachability, and in fact, Büchi objectives, become PSPACE-complete.

More precisely, we show that deciding the existence of $\delta > 0$, and of a strategy for Controller in the conservative perturbation game so as to ensure infinite runs satisfying a given Büchi condition is PSPACE-complete. Thus, this is the complexity of the problem in the exact setting. We characterize "robustly controllable" timed automata, *i.e.*, those in which Controller has a winning strategy, by showing that Controller can win precisely when the timed automaton has an accepting *aperiodic* lasso. Aperiodicity [Sta12] is a variant of *forgetfulness* introduced in [BA11] in a different context, to study the entropy of timed languages. Our characterisation confirms the suggestion of [BA11] that this notion could be significant in the study of robustness. Our results rely on the combination of various techniques used for studying timed automata: Forgetful and aperiodic cycles as considered in [BA11, Sta12], topological semantics of [GHJ97], shrinking from Chapters 5 and 7, and reachability relations of [Pur00].

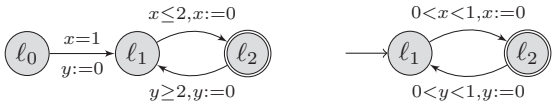
The results presented in this chapter are under submission for publication [SBMR13].

8.2 Robust Büchi Objectives

In this chapter, we consider general timed automata with distinct labels and assume that the clocks are bounded above by a constant. Note that assuming distinct labels is for convenience only and it is no loss of generality since we are not interested in languages, but rather in location-based Büchi objectives.

The problem we are addressing in this chapter is the following.

Definition 8.2.1. *The* parameterized robust controller synthesis in the conservative perturbation game semantics *asks, given a timed automaton $\mathcal{A}$ and a Büchi objective B, whether there exists $\delta > 0$ such that Controller has a winning strategy in $\mathcal{G}_\delta^{\mathsf{cons}}(\mathcal{A})$ for the objective B.*



(a) (b)

Figure 8.1: On the left, a copy of the timed automaton $\mathcal{A}^2$ of Chapter 2 that is not robustly controllable for the Büchi objective $\{\ell_2\}$. In fact, Perturbator can enforce that the value of x be increased by δ at each arrival at ℓ_1, thus blocking the run eventually. On the right, the timed automaton (from [BA11]) is robustly controllable for the Büchi objective $\{\ell_2\}$. In fact, perturbations at a given transition do not affect the rest of the run; they are *forgotten*.

Figure 8.1 shows examples of controllable and uncontrollable timed automata in this sense. The main result of this paper is the following.

Theorem 8.2.2. *Parameterized robust controller synthesis in the conservative perturbation game semantics is* PSPACE*-complete for timed automata with Büchi objectives.*

The next section introduces several notions we need to state our main lemma (Lemma 8.4.1), which characterizes timed automata that are robustly controllable, based on the nature of the lassos of the region automata.

8.3 Regions, Orbit Graphs, Algebra, Topology

We defined regions, region automata, and orbit graphs in Chapter 3. In this section, we give some additional definitions and properties.

Regions and Orbit Graphs We introduce the following definitions on regions. A region r is said *non-punctual* if it contains some $\nu \in r$ such that $\nu + [-\epsilon, \epsilon] \subseteq r$ for some $\epsilon > 0$. It is said *punctual* otherwise. A path $\pi = q_1 e_1 q_2 e_2 \dots q_n$ is *non-punctual* if whenever $e_i = \mathsf{delay}$, q_{i+1} is a non-punctual region.

We will use extensively the orbit graphs defined in Chapter 4. We need the following additional definitions and properties. Any region r has at most one vertex $v \in \mathcal{V}(r)$ such that both v and $v+1$ belong to $\mathcal{V}(r)$. If these exist, then $v = \inf(r)$ and $v + 1 = \sup(r)$. Moreover, $\sup(r) = \inf(r) + 1$ if, and only if r is non-punctual.

Given any path π, for any node (i, v) of $\gamma(\pi)$, let $\mathsf{Succ}((i, v))$ denote the set of nodes $(i+1, w)$ with $\big((i, v), (i+1, w)\big)$ is an edge. We also extend $\mathsf{Succ}(\cdot)$ to sets of nodes. A strongly connected component (SCC) of a graph is *initial* if it is not reachable from any other SCC. We distinguish the following folded orbit graphs.

Definition 8.3.1. *A* forgetful cycle *of $\mathcal{R}(\mathcal{A})$ is a cycle whose folded orbit graph is strongly connected. A cycle π is* aperiodic *if for all $k \geq 1$, π^k is forgetful. A lasso is said to be aperiodic if its cycle is.*

Forgetfulness was recently introduced in [BA11] for studying the entropy of timed languages (note however that their notion requires the graphs to be complete; this is not an important difference, see Lemma 8.6.3). This was further studied in [Sta12] in the context of frequencies in timed automata,

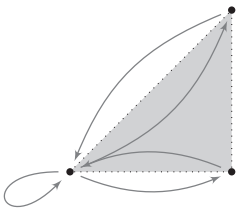
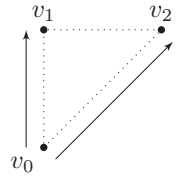


Figure 8.2: The (forgetful) folded orbit graph of the region automaton of the timed automaton of Fig. 8.1(b).

where aperiodicity was defined. Already in these works, forgetful cycles are used to discard convergent runs. As we show in this work, these notions can be used to characterize robust safety along infinite executions in the game semantics.

An example of a non-forgetful cycle was given in Fig. 3.1 in Chapter 4. The timed automaton of Fig. 8.1(b) contains a forgetful cycle, shown in Fig. 8.2.

Some Linear Algebra For any set of vectors, we denote by $\mathsf{Span}(\mathcal{B})$ the linear span of $\mathcal{B}$, *i.e.* the set of linear combinations of $\mathcal{B}$. In the proofs, we will often use the vertices of a region to define a basis of a vector space that contains the region.



Lemma 8.3.2. *Let r be any region, and let $v_0 = \inf(r)$. The set of vectors $\mathcal{B}_{v_0} = \{v - \inf(r)\}_{v \in \mathcal{V}(r) \setminus \{v_0\}}$ is linearly independent. Moreover, the affine space $v_0 + \mathsf{Span}(\mathcal{B}_{v_0})$ contains r.*

Proof. Recall the definition of vertices of r using the partition $X_0, X_1, \ldots, X_m$ of the clocks according to their fractional parts. The vertex $v_0 = \inf(r)$ corresponds to the case where all the clocks are equal to their lower bounds. Region r has one vertex v_i for each $1 \leq i \leq m$, in which all clocks $X_1, \ldots, X_i$ are equal to their lower bounds, and the rest of them are equal to their upper bounds. Hence, one can write $v_i - v_0$ as a column vector of $|X_0| + \ldots + |X_i|$ zeros, followed by $|X_{i+1}| + \ldots + |X_m|$ ones, in the orthogonal basis of $\mathbb{R}^{\mathcal{C}}$. This set is clearly independent.

To see that the affine space $v_0 + \mathsf{Span}(\mathcal{B}_{\inf(r)})$ contains the entire region, write any $\nu \in r$ as the convex combination of the vertices of r: $\nu = \sum_{v \in \mathcal{V}(r)} \lambda_v v$. We get that $\nu - \inf(r) = \sum_{v \in \mathcal{V}(r)} \lambda_v (v - \inf(r))$, since $\sum_{v \in \mathcal{V}(r)} \lambda_v v - v_0 = \sum_{v \in \mathcal{V}(r)} \lambda_v (v - v_0) = \sum_{v \neq v_0} \lambda_v (v - v_0)$, using $\sum_v \lambda_v = 1$. □

Let the dimension of a subset $r \subseteq \mathbb{R}^{\mathcal{C}}$ be the least d such that a affine subspace of $\mathbb{R}^{\mathcal{C}}$ of dimension d contains r.

Lemma 8.3.3. *Let r be a region, and $X_0, X_1, \ldots, X_m$ the partition of the clocks according to their fractional values in r. Then, r has dimension m.*

Proof. Follows immediately from Lemma 8.3.2 since r has $m + 1$ vertices. □

Metric on Timed Traces Since all edge labels are distinct, we will write timed traces as sequences $(t_i, e_i)_{1\leq i\leq n}$ where $t_i \geq 0$ and each e_i is an edge. Given a path π of $\mathcal{R}(\mathcal{A})$, a valuation $\nu \in \mathsf{first}(\pi)$, we say that a timed trace $(t_i, e_i)_{1\leq i\leq n}$ is *feasible* for π and ν if there is a run starting at ν, along π, whose timed trace is $(t_i, e_i)_{1\leq i\leq n}$.

We define a variant of the metric of [GHJ97] on timed traces, which defines the same topology. Given two timed traces $u = (t_i, e_i)_{1\leq i\leq n}$ and $u' = (t'_i, e'_i)_{1\leq i\leq n}$, we let

$$d(u, u') = \infty \text{ if } \mathsf{untime}(u) \neq \mathsf{untime}(u'),$$
$$d(u, u') = \max\{|t_i - t'_i|, 1 \leq i \leq n\} \text{ otherwise.}$$

We define $\mathsf{Ball}_d(u, \epsilon)$ as the open ball of radius ϵ around u in this metric. Note that the original metric was defined using timestamps rather than delays, but both metrics define the same topology [GHJ97].

For a path π, let $\mathsf{ttrace}(\pi)$ denote the set of timed traces that are feasible for π and some valuation in $\mathsf{first}(\pi)$.

Proposition 8.3.4 ([GHJ97]). *If π is a non-punctual path, then $\mathsf{ttrace}(\pi)$ is an open set (for the topology induced by d).*

We will also use the usual d_∞ metric on $\mathbb{R}^{\mathcal{C}}$, defined as

$$d_\infty(\nu, \nu') = \max_{x\in\mathcal{C}} |\nu(x) - \nu'(x)|.$$

We denote open balls in this metric by $\mathsf{Ball}_{d_\infty}(\nu, \epsilon)$.

8.4 Main Lemma

Our main result is based on the following lemma, which gives a characterization of robust timed automata using aperiodic lassos of region automata.

Lemma 8.4.1 (Main Lemma). *For any timed automaton $\mathcal{A}$ and Büchi objective B, Controller has a winning strategy for some $\delta > 0$, if, and only if $\mathcal{R}(\mathcal{A})$ has a reachable aperiodic non-punctual B-winning lasso.*

The algorithm deciding robust safety consists in looking for such lassos. These cycles need not be simple, but Section 8.7 shows how this can be done in polynomial space. We also describe how winning strategies parameterized by δ can be computed. The two directions of the main lemma are proved using different techniques; they are presented respectively in Sections 8.5 and 8.6.

8.5 No Aperiodic Lassos Implies No Robustness

In this section, we prove that Controller loses if there is no aperiodic winning lassos. The idea is that if no accepting lasso of $\mathcal{R}(\mathcal{A})$ is aperiodic, then, as we show, the projection of any play to $\mathcal{R}(\mathcal{A})$ eventually enters and stays in a non-forgetful cycle. Then, we choose an appropriate *Lyapunov function* $L_I(\cdot)$ defined on valuations and taking nonnegative values, and describe a strategy for Perturbator such that the value of $L_I(\cdot)$ is decreased by at least ϵ at each iteration of the cycle. Hence, Controller cannot cycle infinitely on such cycles: either it reaches a deadlock, or it cycles on non-accepting lassos. In the rest of this section, we describe Perturbator's strategy, prove some properties on its outcomes, choose a function $L_I(\cdot)$, and finally prove the first direction of the main lemma.

8.5.1 Reachability Relations

We already noted that any valuation ν can be written as the convex combination of the vertices of its region, i.e. $\nu = \sum_{v \in \mathcal{V}(\mathsf{reg}(\nu))} \lambda_v v$ for some unique coefficients $\lambda_v \geq 0$ with $\sum_v \lambda_v = 1$. When the region is clear from context, we will simply write $\nu = \lambda \mathbf{v}$. Given a path π and a vertex v of $\mathsf{first}(\pi)$, let us denote by $R_{\Gamma(\pi)}(v)$ the set of nodes $w \in \mathcal{V}(\mathsf{last}(\pi))$ such that $(v, w) \in E(\Gamma(\pi))$. Thus, this is the "image" of v by the path π. Puri showed in [Pur00] that the reachability along paths can be characterized using orbit graphs.

Lemma 8.5.1 ([Pur00])**.** *Let π be a path from region r to s. Consider any $\nu \in r$ with $\nu = \sum_{v \in \mathcal{V}(r)} \lambda_v v$ for some coefficients $\lambda_v \geq 0$ and $\sum \lambda_v = 1$. If $\nu \xrightarrow{\pi} \nu'$, then for each $v \in \mathcal{V}(r)$, there exists a probability distribution $\{p_{v,w}^{\nu,\nu'}\}_{w \in R_{\Gamma(\pi)}(v)}$ over $R_{\Gamma(\pi)}(v)$ such that*

$$\nu' = \sum_{v \in \mathcal{V}(r)} \lambda_v \sum_{w \in R_{\Gamma(\pi)}(v)} p_{v,w}^{\nu,\nu'} w. \tag{8.1}$$

The converse holds for $\bar{\pi}$: If there exist probability distributions $p_{v,w}^{\nu,\nu'}$ satisfying (8.1), then $\nu \xrightarrow{\bar{\pi}} \nu'$.

This lemma shows that any successor of a point $\nu = \sum_i \lambda_i v_i$ can be obtained by distributing each weight λ_i of any vertex v_i to its successors following a probability distribution.

Example 8.5.2. *The automaton of Fig. 8.1(a) contains a cycle on the region $r = [\![1 < x, y < 2 \wedge 0 < x - y < 1]\!]$. The vertices of r are $v_1 = (1, 0), v_2 = (2, 0), v_3 = (2, 1)$. Figure 8.3 shows the folded orbit graph given with a probability distribution on the outgoing edges of each node. Consider a point $\nu = \frac{1}{3}v_1 + \frac{1}{3}v_2 + \frac{1}{3}v_3$. Then, Lemma 8.5.1 says that $\nu' = \sum_{1 \leq i \leq 3} \lambda_i v_i$ is reachable from ν along the cycle, where $\lambda_1 = \frac{1}{3}0.5 + \frac{1}{3}0.4 = \frac{9}{30}$, $\lambda_2 = \frac{1}{3}1 + \frac{1}{3}0.3 = \frac{13}{30}$, and $\lambda_3 = \frac{1}{3}0.6 + \frac{1}{3}0.2 = \frac{4}{15}$.*

Fig. 8.3 also shows in gray the strongly connected components of the graph. The component I is an initial one. Notice that L_I is indeed decreasing in this example: $\frac{1}{3} + \frac{1}{3} \geq \frac{9}{30} + \frac{4}{15}$.

The following lemma is a re-statement of Lemma 8.5.1 as suggested in [BA11]. For a folded orbit graph $\Gamma(\pi)$, $M(\Gamma(\pi))$ denotes the *adjacency matrix*, of size $n \times n$, if the graph has n nodes, defined as follows. $(M(\Gamma(\pi)))_{u,v} = 1$ if, and only if there is an edge from v to u, and $(M(\Gamma(\pi)))_{u,v} = 0$ otherwise. A *stochastic matrix* is a matrix in which the sum of each column equals 1. We see λ as a column vector.

Lemma 8.5.3 ([BA11])**.** *Let π be a path from region r to s. Let $\nu = \sum_{v \in \mathcal{V}(r)} \lambda_v v$ and $\nu' = \sum_{u \in \mathcal{V}(s)} \lambda'_u u$. Then, $\nu \xrightarrow{\bar{\pi}} \nu'$ if, and only if there is a stochastic matrix $P \leq M(\Gamma(\pi))$ such that $\lambda' = P\lambda$.*

In the above lemma, the inequality between matrices is to be interpreted componentwise. Thus, the matrix P is simply (the transpose of) the matrix of the probabilities $p_{v,w}^{\nu,\nu'}$ of Lemma 8.5.1.

For any region r, and any subset $I \subseteq \mathcal{V}(r)$, we define the function $L_I : \bar{r} \to \mathbb{R}_{\geq 0}$ as,

$$L_I(\nu) = \sum_{v \in I} \lambda_v, \quad \text{where } \nu = \lambda \mathbf{v}.$$

It is shown in [BA11] that given any cycle π, if I is chosen as the initial strongly connected component of $\Gamma(\pi)$, then for any run $\nu \xrightarrow{\pi} \nu'$, $L_I(\nu') \leq L_I(\nu)$. We will abusively use $L_I(\cdot)$ for a subset I of nodes of $\gamma(\pi)$ or $\Gamma(\pi)$, that correspond to a same region. Notice that $0 \leq L_I(\cdot) \leq 1$.

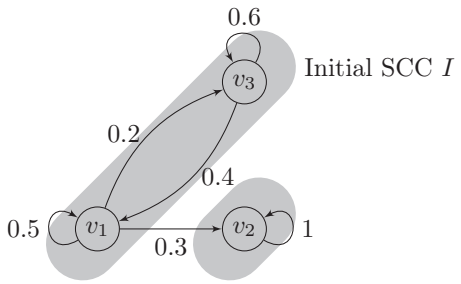


Figure 8.3: The folded orbit graph of the cycle of Fig. 3.1, augmented with a probability distribution.

8.5.2 A global strategy for Perturbator

Let us call a valuation v *ϵ-far* if $v + [-\epsilon, \epsilon] \subseteq \mathsf{reg}(v)$. A run is ϵ-far if all delays end in ϵ-far valuations. We define a global strategy $\sigma_\delta^{\mathsf{P}}$ for Perturbator that ensures ϵ-far runs, as follows. After any delay $\nu \xrightarrow{d} \nu'$, chosen by Controller, consider the regions spanned by the set $\nu' + [0, \delta]$. It is easy to see that this set intersects at most $|\mathcal{C}| + 1$ different regions, all of which must satisfy the guard by definition of the game. So some region r satisfies $\nu' + [\alpha, \beta] \subseteq r$, for some $0 \leq \alpha < \beta \leq \delta$ with $\beta - \alpha \geq \frac{\delta}{|\mathcal{C}|+1}$. The strategy $\sigma_\delta^{\mathsf{P}}$ consists in choosing the perturbation as $\frac{1}{2}(\beta - \alpha)$. This guarantees time progress (of at least $\frac{\delta}{2(|\mathcal{C}|+1)}$). Moreover, the resulting valuation is always ϵ-far in its region, where $\epsilon = \frac{\delta}{2(|\mathcal{C}|+1)}$.

We study the properties of the runs $\mathsf{Outcome}_{\mathcal{A}}^{\delta}(\cdot, \sigma_\delta^{\mathsf{P}})$. The following proposition is a key element of the proof. Using the ϵ-far property of the runs, this proposition derives a bound on the convex combination coefficients of all visited valuations.

Proposition 8.5.4. *Let $\rho \in \mathsf{Outcome}_{\mathcal{A}}(\cdot, \sigma_\delta^{\mathsf{P}})$. For any $i \geq 1$, if we write $\mathsf{state}_i(\rho) = \lambda \mathbf{v}$, then $\lambda_v \geq \epsilon$ for all vertices $v \in \mathcal{V}(\mathsf{reg}(\mathsf{state}_i(\rho)))$.*

We begin with simple properties of non-punctual regions.

Lemma 8.5.5. *If π is a non-punctual delay from region r_1 to r_2, then exactly one of the following cases must hold for the graph $\gamma(\pi)$:*

1. *Exactly one vertex of r_1 has outdegree 2. The successors of this vertex are $\inf(r_2)$ and $\sup(r_2)$. All other vertices of r_1 have outdegree 1.*
2. *Exactly two vertices of r_1 have outdegree 2. These vertices are $\inf(r_1)$ and $\sup(r_1)$ and both lead to $\inf(r_2)$ and $\sup(r_2)$. All other vertices of r_1 have outdegree 1.*

Moreover all vertices of r_2 have indegree at most 2, and the predecessors of any vertex of indegree 2 are $\inf(r_1)$ and $\sup(r_1)$.

We note the following lemma on ϵ-far valuations.

Lemma 8.5.6. *If ν is an ϵ-far valuation, then for all clock x, if l_x and u_x denote respectively the lower and upper bounds on x in region $\mathsf{reg}(\nu)$, then $l_x + \epsilon \leq \nu(x) \leq u_x - \epsilon$. It follows that for all $v \in \mathcal{V}(\mathsf{reg}(\nu))$, $d_\infty(v, \nu) \geq \epsilon$.*

We show the proposition first for discrete transitions, then for ϵ-far delays. The following lemma treats the clock resets, thus discrete transitions. We denote by $\min(\lambda)$ the minimum among the components of the vector λ.

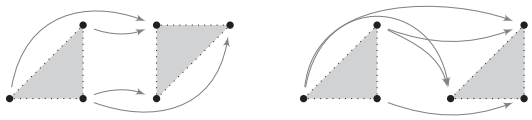
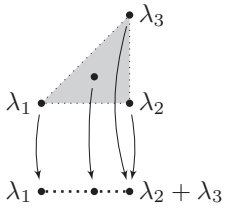


Figure 8.4: Illustration of both cases of Lemma 8.5.5

Lemma 8.5.7. *For any $\nu = \lambda\mathbf{v}$ and $\nu' = \lambda'\mathbf{v}'$ with $\nu' = \nu[R \leftarrow 0]$ for some $R \subseteq \mathcal{C}$, $\min(\lambda') \geq \min(\lambda)$.*

Proof. The orbit graph of a clock reset has the property that all vertices of $\mathsf{reg}(\nu)$ has outdegree 1 (since the reset defines a function). By Lemma 8.5.1, each $\lambda_{v'_i}$ is equal to the sum of a subset of the components of $\mathbf{v}$, namely those vertices that have an edge to vertex v'_i. The inequality follows. □

Proving Proposition 8.5.4 for delays is more difficult and is dealt with in the following lemma. We write $\lambda \geq \epsilon$ to mean that all components of the vector satisfy the inequality.



Lemma 8.5.8. *Let $\nu = \lambda\mathbf{v}$ and $\nu' = \lambda'\mathbf{v}'$ where $\nu' = \nu + d$ for some $d \geq 0$ and ν' is ϵ-far. If $\mathsf{reg}(\nu)$ is a punctual region, and $\lambda \geq \epsilon$, then $\lambda' \geq \epsilon$.*

Proof. We denote $r = \mathsf{reg}(\nu)$ and $s = \mathsf{reg}(\nu')$. Let γ denote the orbit graph of the delay $r \to s$. Because we have an ϵ-far delay, s must have vertices $\inf(s) \neq \sup(s)$.

We first show that $\lambda'_{\inf(s)}, \lambda'_{\sup(s)} \geq \epsilon$. There exists a clock x such that $\sup(s)(x) = v(x) + 1$ for any other vertex v of region s (x is one of the first clocks that appear in the fractional ordering of the clocks inside s). Let us denote by $l = \inf(s)(x)$ and $u = \sup(s)(x)$. By Lemma 8.5.6, we must have

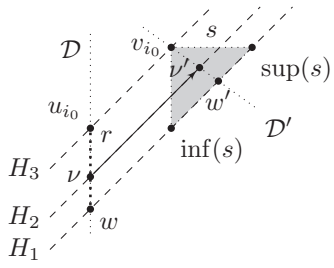
$$l + \epsilon \leq \lambda'_{\sup(s)} \cdot u + \sum_{v \neq \sup(s)} \lambda'_v \cdot v(x).$$

Using the fact that $u = l + 1$, and $v(x) = l$ for all $v \neq \sup(s)$, we get

$$\begin{array}{rl} l + \epsilon & \leq \lambda'_{\sup(s)} \cdot u + \sum_{v \neq \sup(s)} \lambda'_v \cdot l, \\ l + \epsilon & \leq \lambda'_{\sup(s)} + l \sum_v \lambda'_v, \\ \epsilon & \leq \lambda'_{\sup(s)}. \end{array}$$

A symmetric argument can be used to prove $\lambda'_{\inf(s)} \geq \epsilon$.

Consider now a vertex $v_{i_0} \notin \{\sup(s), \inf(s)\}$. We define the vector space $\mathcal{E}$ generated by the basis $\{v - \inf(s)\}_{v \neq \inf(s)}$. We consider the linear function $f_{i_0} : \mathcal{E} \mapsto \mathbb{R}$ which extracts from a point its coefficient of the vector $(v_{i_0} - \inf(s))$. Write $\mathcal{E}' = f_{i_0}^{-1}(0)$, a hyperplane in $\mathcal{E}$. Then, $H_1 = \inf(s) + \mathcal{E}'$ is an affine hyperplane in $\inf(s) + \mathcal{E}$. Observe that for points in s, the coefficients under this basis corresponds precisely to the coefficients of the convex combination of the vertices. Thus, the set $H_1 \cap s$ contains precisely those valuations of s whose convex combination assigns 0 to vertex v_{i_0}. In particular, $\mathcal{E}'$ contains the line of unit direction $\mathbf{1}$ since it contains $[\inf(s), \sup(s)]$. Thus, H_1



contains all time-predecessors of all points of $H_1 \cap \bar{s}$, and in particular for all vertices $v \neq v_{i_0}$ of s, the predecessors of v in γ belong to H_1.

Consider the unique predecessor u_{i_0} of v_{i_0} in γ (it is unique because $\mathsf{reg}(\nu)$ is punctual). We know that $u_{i_0} \notin H_1$ since otherwise $v_{i_0} \in H_1$. There exist affine hyperplanes H_2 and H_3 parallel to H_1, containing respectively the segments $[\nu, \nu']$ and $[u_{i_0}, v_{i_0}]$. In fact, both segments have direction $\mathbf{1}$, so one can simply define H_2 and H_3 as translations of H_1. Let $\mathcal{D}$ denote the line containing ν and u_{i_0}, and $\mathcal{D}'$ the line containing ν' and v_{i_0}. These lines are not parallel to H_1. In fact, both ν and ν' are in the interior of their regions, so we have $0 < \lambda < 1$ and $0 < \lambda' < 1$. Now, if $\mathcal{D}'$ is parallel to H_3, because $v_{i_0} \in H_3 \cap \mathcal{D}'$, we would have $\nu' \in H_3$. But $f_{i_0}(\nu' - v_{i_0}) \neq 0$ since $0 < \lambda' < 1$, which is a contradiction. Similarly, if $\mathcal{D}$ were parallel to H_3, then $\nu \in H_3$. In this case, for some delay $d > 0$, $\nu + d$ could be in the interior of s, which would lead to a contradiction as above.

Let $w = H_1 \cap \mathcal{D}$ and $w' = H_1 \cap \mathcal{D}'$, which are therefore points. By the Intercept Theorem (below), we have

$$\frac{|u_{i_0} - \nu|}{|u_{i_0} - w|} = \frac{|v_{i_0} - \nu'|}{|v_{i_0} - w'|}. \tag{8.2}$$

On the other hand, we have

$$\nu = \Big(\sum_{u \neq u_{i_0}} \lambda_u \Big) w_1 + \lambda_{u_{i_0}} u_{i_0},$$

where w_1 is defined as the convex combination of all vertices but u_{i_0} using coefficients $\frac{\lambda_u}{\sum_{u \neq u_{i_0}} \lambda_u}$ (which is well-defined as $\lambda \geq \epsilon$). Let w_2 be a time successor of w_1 in the closure of $\mathsf{reg}(s)$. It can be defined as a barycenter of all vertices of $\mathsf{reg}(s)$ except v_{i_0}. It therefore holds that $f_{i_0}(w_2) = 0$, and hence that both w_1 and w_2 are in H_1. Also $w_1 \in \mathcal{D}$ by the above equation. Therefore $w_1 = w$. We have similarly that

$$\nu' = \Big(\sum_{v \neq v_{i_0}} \lambda_v \Big) w_1' + \lambda_{v_{i_0}} v_{i_0},$$

with $w_1' = w'$. By (8.2) and the fact that $1 - \lambda_{v_{i_0}} = \sum_{v \neq v_{i_0}} \lambda_v$, we have $\lambda_{v_{i_0}} = \lambda_{u_{i_0}}$, therefore $\lambda_{v_{i_0}} \geq \epsilon$. □

The Intercept Theorem is the generalization to arbitrary finite dimension of a simple theorem we learned in school.

Theorem 8.5.9 (Intercept Theorem)**.** *Let d and d' be lines in an affine space. If there are three parallel hyperplanes intersecting d and d' respectively at points A, A', B, B', C, C', Then, $\frac{|AB|}{|AC|} = \frac{|A'B'|}{|A'C'|}$.*

Proof of Proposition 8.5.4. By induction on the length of a run under strategy $\sigma_\delta^{\mathsf{P}}$. Initially, we are in a singleton region with one vertex, so the only convex coefficient is 1. In the induction step, resets follow from Lemma 8.5.7. Delays follow from Lemma 8.5.8 since all delays are ϵ-far, and by the fact that the valuation after the latest reset lies in a punctual region. □

We prove a similar proposition for the edge probabilities of Lemma 8.5.3. Proposition 8.5.10 says that all edges of the folded orbit graph receive a probability of at least $\Omega > 0$ along ϵ-far delays, according to the interpretation of Lemma 8.5.3.

Proposition 8.5.10. *Let $\nu = \lambda\mathbf{v}$ and $\nu' = \lambda'\mathbf{v}'$ denote two valuations satisfying $\lambda, \lambda' \geq \epsilon$, and such that $\nu \xrightarrow{\pi} \nu'$ is an ϵ-far delay of duration at least ϵ. Then, there exists a stochastic matrix P such that*

$$\Omega M(\Gamma(\pi)) \leq P \leq M(\Gamma(\pi)),$$

such that $\lambda' = P\lambda$, where $\Omega = \min(\frac{1}{2}, \frac{\epsilon}{2})$.

Proof. Let r and s denote the source and target regions.

Assume that both $\inf(s)$ and $\sup(s)$ have indegree 1, and let u be a predecessor. We have,

$$\lambda'_{\inf(s)} = \lambda_u \cdot p_{u,\inf(s)},$$
$$\lambda'_{\sup(s)} = \lambda_u \cdot p_{u,\sup(s)}.$$

If u is the only vertex of r, then $\lambda_u = 1$, and we get that $p_{u,\inf(s)}, p_{u,\sup(s)} \geq \epsilon$ since $\lambda'_{\inf(s)}, \lambda'_{\sup(s)} \geq \epsilon$. Otherwise, we must have $\lambda_u \leq 1 - \epsilon$. Again, because $\lambda'_{\inf(s)}, \lambda'_{sup(s)} \geq \epsilon$, we get that $p_{u,\inf(s)}, p_{u,\sup(s)} \geq \frac{\epsilon}{1-\epsilon} \geq \Omega$. All other edges have outdegree 1 so the probabilities are equal to 1.

Assume now that $\sup(s)$ has indegree 2, but $\inf(s)$ has indegree 1. We show that in this case $r = s$. Observe that $\inf(s)$ cannot have indegree higher than that of $\sup(s)$ since any predecessor of $\inf(s)$ is a predecessor of $\sup(s)$. Now, $\sup(s)$ cannot have both its predecessors outside $\mathcal{V}(s)$ since otherwise these would be predecessors of $\inf(s)$. Thus, $\inf(s) \in \mathcal{V}(r)$, and is a predecessor of $\sup(s)$. In this case, $\inf(s)$ is a predecessor of itself since this is a delay transition. Furthermore, if $\sup(s)$ has its other predecessor outside s, this would again be a predecesor of $\inf(s)$, and $\inf(s)$ would have indegree 2. Therefore, the predecessors of $\sup(s)$ must be $\sup(s)$ and $\inf(s)$, and the predecessor of $\inf(s)$ is itself. This is only possible when $s = r$.

We have,

$$\lambda'_{\inf(s)} = \lambda_{\inf(s)} \cdot p_{\inf(s),\inf(s)},$$
$$\lambda'_{\sup(s)} = \lambda_{\inf(s)} \cdot p_{\inf(s),\sup(s)} + \lambda_{\sup(s)} \cdot p_{\sup(s),\sup(s)}.$$

We get that $p_{\inf(s),\inf(s)} \geq \Omega$. Here, $p_{\sup(s),\sup(s)} = 1$ since $\sup(s)$ has outdegree 1 by assumption. Let x denote the clock with the least fractional value in s. Notice that the clock x has the same value in all vertices apart from $\sup(s)$. We have $\nu(x) = \inf(s)(x) \cdot (1 - \lambda_{\sup(s)}) + \sup(s)(x) \cdot \lambda_{\sup(s)}$, which gives $\nu(x) - \inf(s)(x) = \lambda_{\sup(s)}$. Similarly, we have $\nu'(x) - \inf(s)(x) = \lambda'_{\sup(s)}$. Since the delay is at least ϵ, we have

$$(\nu'(x) - \inf(s)(x)) - (\nu(x) - \inf(s)(x)) = \nu'(x) - \nu(x) \geq \epsilon.$$

which is equivalent to $\lambda'_{\sup(s)} - \lambda_{\sup(s)} \geq \epsilon$. By writing $\lambda'_{\sup(s)}$ as above, this is equivalent to

$$\lambda_{\inf(s)} \cdot p_{\inf(s),\sup(s)} + \lambda_{\sup(s)} \cdot (p_{\sup(s),\sup(s)} - 1) \geq \epsilon.$$

Thus, we must have $\lambda_{\inf(s)} \cdot p_{\inf(s),\sup(s)} \geq \epsilon$, which means that $p_{\inf(s),\sup(s)} \geq \frac{\epsilon}{1-\epsilon} \geq \epsilon$, as required.

Let us now consider the case where two vertices of r have outdegree 2. This means that $\inf(r)$ and $\sup(r)$ are different, and that they have an edge leading to both $\inf(s)$ and $\sup(s)$. To simplify notations, let us write $r_1 = \inf(r), r_2 = \sup(r)$ and similarly s_1 and s_2 for s. We have the following relation.

$$\begin{array}{ll} \lambda'_{s_1} & = p_{11}\lambda_{r_1} + p_{21}\lambda_{r_2}, \\ \lambda'_{s_2} & = p_{12}\lambda_{r_1} + p_{22}\lambda_{r_2}, \end{array} \tag{8.3}$$

where $p_{11}, p_{12}, p_{21}, p_{22} \geq 0$ are given by Lemma 8.5.3 and satisfy $p_{11} + p_{12} = 1$, $p_{21} + p_{22} = 1$. Now, all p_{ij} may not be greater than Ω here. But we will show that one can then modify these to obtain components greater than Ω. In other terms, some stochastic matrix P satisfyin $P \geq \Omega M(\Gamma(\pi))$ will also satisfy Lemma 8.5.3. Without loss of generality, let us assume that $p_{11} < \Omega$. We would like to increase p_{11} by some $\alpha \geq 0$ and still satisfy the above equation, the stochasticity, and obtain a new matrix of p_{ij} whose all components are in $[\Omega, 1-\Omega]$. Then, the new equation is the following:

$$\begin{array}{ll} \lambda'_{s_1} & = (p_{11} + \alpha)\lambda_{r_1} + (p_{21} - \frac{\alpha\lambda_{r_1}}{\lambda_{r_2}})\lambda_{r_2}, \\ \lambda'_{s_2} & = (p_{12} - \alpha)\lambda_{r_1} + (p_{22} + \frac{\alpha\lambda_{r_1}}{\lambda_{r_2}})\lambda_{r_2}. \end{array} \tag{8.4}$$

This equation is satisfied for any α, assuming (8.3) is satisfied. In order to obtain the matrix we are looking for, it suffices to find α such that the following constraints are satisfied (since stochasticity is already satisfied).

$$\begin{array}{c} \Omega \leq p_{11} + \alpha \leq 1 - \Omega, \\ \Omega \leq p_{21} - \frac{\alpha\lambda_{r_1}}{\lambda_{r_2}} \leq 1 - \Omega. \end{array} \tag{8.5}$$

We now use Fourier-Motzkin elimination to show that there is always such a choice for α. Let us rewrite the above constraints on α by separating lower bounds and upper bounds.

$$\Omega - p_{11} \leq \alpha, \tag{8.6}$$

$$\frac{\lambda_{r_2}}{\lambda_{r_1}} p_{21} + \frac{\lambda_{r_2}}{\lambda_{r_1}}(1 + \Omega) \leq \alpha, \tag{8.7}$$

$$\alpha \leq 1 - \Omega - p_{11}, \tag{8.8}$$

$$\alpha \leq p_{21}\frac{\lambda_{r_2}}{\lambda_{r_1}} - \Omega\frac{\lambda_{r_2}}{\lambda_{r_1}}. \tag{8.9}$$

It suffices to show that each pair of lower and upper bounds is satisfied by some α.

(8.6)-(8.8): holds since $\Omega < \frac{1}{2}$.

(8.6)-(8.9): This can be written as $\Omega \leq \frac{\lambda_{r_2}}{\lambda_{r_1}+\lambda_{r_2}} p_{21} + \frac{\lambda_{r_1}}{\lambda_{r_1}+\lambda_{r_2}} p_{11}$. But $\epsilon \leq \lambda_{r_2} p_{21} + \lambda_{r_1} p_1 1$ since this is equal to λ'_{s_1}. Since $\lambda_{r_1} + \lambda_{r_2} \leq 1$ and $\Omega \leq \epsilon$, the constraint holds.

(8.7)-(8.8): This is equivalent to

$$\begin{array}{ll} \Omega & \leq \frac{\lambda_{r_1}}{\lambda_{r_1}+\lambda_{r_2}}(1 - p_{11}) + \frac{\lambda_{r_2}}{\lambda_{r_1}+\lambda_{r_2}}(1 - p_{21}) \\ & = 1 - \left(\frac{\lambda_{r_1}}{\lambda_{r_1}+\lambda_{r_2}} p_{11} + \frac{\lambda_{r_2}}{\lambda_{r_1}+\lambda_{r_2}} p_{21}\right). \end{array}$$

We have $\frac{\lambda_{r_1}}{\lambda_{r_1}+\lambda_{r_2}} p_{11} + \frac{\lambda_{r_2}}{\lambda_{r_1}+\lambda_{r_2}} p_{21} \leq \frac{\lambda_{r_1}}{\lambda_{r_1}+\lambda_{r_2}}\Omega + \frac{\lambda_{r_2}}{\lambda_{r_1}+\lambda_{r_2}}$ since $p_{11} < \Omega$ and $p_{21} \leq 1$. So, it suffices to show that $\Omega \leq 1 - \frac{\lambda_{r_1}}{\lambda_{r_1}+\lambda_{r_2}}\Omega - \frac{\lambda_{r_1}}{\lambda_{r_1}+\lambda_{r_2}}$, *i.e.* $\Omega \leq \frac{\lambda_{r_1}}{2\lambda_{r_1}+\lambda_{r_2}}$. But this holds since $\epsilon \leq \lambda_{r_1}$ and $2\lambda_{r_1} + \lambda_{r_2} \leq 2 - \epsilon$. In fact, we have $\Omega \leq \frac{\epsilon}{2-\epsilon}$.

(8.7)-(8.9): This is equivalent to $2\frac{\lambda_{r_2}}{\lambda_{r_1}}\Omega \leq \frac{\lambda_{r_2}}{\lambda_{r_1}}$ and holds since $\Omega \leq \frac{1}{2}$. □

8.5.3 Decreasing Lyapunov function

In the previous paragraph, we established lower bounds on the convex coefficients of the visited valuations. We use this property to find a Lyapunov function that strictly decreases at each cycle. Let us first note the following property of the functions $L_I(\cdot)$.

Lemma 8.5.11. *Let π be a cycle, and write $\gamma(\pi) = (V_1 \cup \ldots \cup V_n, f_G, E)$ for its orbit graph, and $\Gamma(\pi) = (V, f'_G, E')$ for its folded orbit graph. Let $I_1 \subseteq V$ be any subset for which $(u, v) \in E'$ with $v \in I_1$ implies $u \in I_1$. Define $I_i = \mathsf{Succ}_{\gamma(\pi)}(I_{i-1})$ for all $2 \leq i \leq n$. Then, for any run ρ along π, we have $L_{I_i}(\mathsf{state}_i(\rho)) = L_{I_1}(\mathsf{state}_1(\rho))$ for all $1 \leq i \leq n$.*

Proof. By induction, using Lemma 8.5.1. □

The following lemma shows that runs along non-punctual progress cycles can be modified so as to reach any valuation in a ball around the target state. This gives the dimension of the set of valuations reachable along a progress cycle starting from a given valuation. The result and its proof are similar to [DDMR08, Lemma 29].

Lemma 8.5.12. *Let π be a non-punctual progress cycle, and $(\ell, \nu) \xrightarrow{\pi} (\ell, \nu')$ a run along π. Then, there exists $\epsilon > 0$ such that there exists a run from (ℓ, ν), along π, to any point in $\{\ell\} \times (\mathsf{Ball}_{d_\infty}(\nu', \epsilon) \cap \mathsf{reg}(\nu'))$.*

Proof. Consider the partition of the clocks inside $\mathsf{first}(\pi)$, according to their fractional ordering. One can factor π as $\pi = \pi' \pi_n \pi_{n-1} \ldots \pi_1$, where each π_i starts with the latest reset of some clocks $Y_i \subseteq \mathcal{C}$, with $Y_i = X_j$ for some j, and π_1 ends with a reset of the clocks X_0 if this set is non-empty. To see that for any i, there exists j such that $Y_i = X_j$, suppose the latest resets of some clocks x and y are at different transitions, say, $\mathsf{state}_i(\pi)$ and $\mathsf{state}_j(\pi)$ respectively, with $i < j$. Then, they belong to the same set X_k if, and only if the total delay of any run along $\pi_{i \ldots j}$ is an integer. But this is only possible if equality constraints are used, which is not the case since π is non-punctual. Moreover, we have $n = m$, since all clocks are reset at least once as π is a progress cycle.

Let us factorize the run ρ as $\rho = \rho' \rho_m \rho_{m-1} \ldots \rho_1$ where each ρ_i is along π_i and ρ' along π'. Then, for each clock $x \in X_i$, $\nu'(x) = \mathsf{delay}(\rho_i \rho_{i-1} \ldots \rho_1)$, where $\mathsf{delay}(\cdot)$ denotes the sum of the delays of a given run. Consider the timed trace $(d_i, e_i)_i$ of ρ. We have $d_i > 0$ for all i by definition. By Proposition 8.3.4, any timed trace $(d'_i, e_i)_i$, with $|d_i - d'_i| \leq \epsilon$ for all $i \geq 1$, defines a run starting at (ℓ, ν) along π, if ϵ is chosen small enough. It remains to show that any point $\nu'' \in \mathsf{Ball}_{d_\infty}(\nu', \epsilon) \cap \mathsf{reg}(\nu')$ is reachable from (ℓ, ν) following such a "close" timed trace.

Consider such an ϵ, and any $\nu'' \in \mathsf{reg}(\nu')$ such that $d_\infty(\nu', \nu'') \leq \frac{\epsilon}{2}$. Notice that we have $\mathsf{frac}(\nu''(x)) = \mathsf{frac}(\nu''(y))$, and $\mathsf{frac}(\nu'(x)) = \mathsf{frac}(\nu'(y))$ for any $x, y \in Y_i$, which implies $\nu''(x) - \nu'(x) = \nu''(y) - \nu'(y)$. Thus, we define $(\eta_1, \ldots, \eta_m) \in [-\frac{\epsilon}{2}, \frac{\epsilon}{2}]^m$ such that $\nu'(x) + \eta_i = \nu''(x)$ for all $x \in Y_i$. A run ending in (ℓ, ν'') can be defined as $o = o' o_m \ldots o_1$, by modifying the trace $(d_i, e_i)_i$ so as to satisfy the following properties:

$$
\begin{aligned}
&\mathsf{delay}(o_1) = \mathsf{delay}(\rho_1) + \eta_1,\\
&\mathsf{delay}(o_2) = \mathsf{delay}(\rho_2) + \eta_2 - \eta_1,\\
&\ldots\\
&\mathsf{delay}(o_m) = \mathsf{delay}(\rho_m) + \eta_m - \eta_{m-1},\\
&\mathsf{delay}(o') = \mathsf{delay}(\rho').
\end{aligned}
$$

It is possible to obtain o by modifying an arbitrary delay in each ρ_i since $|\eta_i - \eta_{i+1}| \leq \epsilon$. □

We now prove that the folded orbit graphs of non-punctual progress cycles are always connected. If the cycle is non-forgetful, there are at least two connected SCCs (Corollary 8.5.14).

Lemma 8.5.13. *The folded orbit graph of any non-punctual progress cycle is connected.*

Proof. Consider a non-punctual progress cycle and let us denote $r = \mathsf{first}(\pi)$. Consider the basis $\mathcal{B}_{v_0}$, with $v_0 = \inf(r)$, defined in Lemma 8.3.2. Assume that the orbit graph has at least two disjoint connected components G_1 and G_2. Note that we do not require G_i's to be *strongly* connected components. These are simply two disjoint maximal subgraphs.

Let $v_0 + u$ be a point in r, with $u \in \mathsf{Span}(\mathcal{B}_{v_0})$ and write $u = \sum_v \lambda_v v$. Let f denote the linear function that associates to each element of $\mathsf{Span}(\mathcal{B}_{v_0})$ the sum of the coefficients of the vertices of G_1. Then $f(u) = \sum_{v \in G_1} \lambda_v$. For any run $(v_0 + u) \xrightarrow{\pi} (v_0 + u')$, where $u' = \sum_v \lambda'_v v$, we have, by Lemma 8.5.11, $f(u) = f(u')$. Thus, the set $U \subseteq r$ of valuations reachable from $v_0 + u$ along π lies in the set $v_0 + f^{-1}(f(u))$, which is an affine hyperplane of $v_0 + \mathsf{Span}(\mathcal{B}_{v_0})$. So it has smaller dimension than r. On the other hand, by Lemma 8.5.12, the set U includes the intersection of a d-open ball intersected with r, hence have the same dimension as r. Contradiction. □

Corollary 8.5.14. *The folded orbit graph of a non-punctual non-forgetful progress cycle π contains at least two strongly connected components that are connected. We associate with each π an initial SCC of $\Gamma(\pi)$, which we denote by $I(\pi)$.*

Hence, for any non-punctual non-forgetful cycle π, we consider the function $L_{I(\pi)}$. The following lemma shows that $L_{I(\pi)}$ decreases by at least a fixed amount at each iteration of such a cycle under Perturbator's strategy $\sigma^{\mathsf{P}}_\delta$.

Lemma 8.5.15. *Let $\omega \in \mathit{Outcome}^{\delta}_{\mathcal{A}}(\cdot, \sigma^{P}_{\delta})$, and ρ be a finite prefix of ω such that π, the projection of ρ to regions, is a cycle. If π is a non-forgetful progress cycle, then, writing $\mathit{first}(\rho) = \lambda \mathbf{v}$ and $\mathit{last}(\rho) = \lambda' \mathbf{v}'$, we have,*

$$\sum_{i \in I(\pi)} \lambda'_i \leq \sum_{i \in I(\pi)} \lambda_i - \epsilon^2/2.$$

Proof. We know that π is non-punctual by definition of $\sigma^{\mathsf{P}}_\delta$. Let us write $\gamma(\pi) = (V_1 \cup \ldots \cup V_n, f_G, E)$. For each $1 \leq i \leq n$, let I_i denote the set of nodes of V_i that are reachable from the nodes $\{1\} \times I(\pi) \subseteq V_1$. Observe that $I_1 = \{1\} \times I(\pi)$, and $\{n\} \times I(\pi) \subseteq I_n$ since $I(\pi)$ is a strongly connected component. However, $I_n \neq \{n\} \times I(\pi)$ because of Corollary 8.5.14. Let J_i denote the set of nodes of V_i from which the set $\{n\} \times I(\pi)$ is reachable. We have $J_n = \{n\} \times I(\pi)$ by definition. Because $I(\pi)$ is an initial component, we have $J_i \subseteq I_i$ for all i. Indeed, assume $v \in J_i \setminus I_i$ for some i. Then some node of $\{n\} \times I(\pi)$ is reachable from v. Since each node has at least one predecessor in $\gamma(\pi)$, v is reachable from some node u of V_1. We have that $u \notin I_1$, since otherwise $v \in I_i$. Thus, in $\Gamma(\pi)$, there is an edge from the SCC of u to $I(\pi)$, contradicting the fact that $I(\pi)$ is an initial SCC. Moreover, we have $I_1 = J_1$. In fact, J_n is reachable from any node of I_1 since this is a strongly connected component.

Let i_0 be the least index such that $I_i \neq J_i$. Notice that we have $L_{I_i}(\mathsf{state}_i(\rho)) = L_{I_1}(\mathsf{state}_1(\rho))$ for any $1 \leq i \leq i_0 - 1$, by Lemma 8.5.1 (in fact, all predecessors of I_i are in I_{i-1}, and all nodes of I_{i-1} have at least one successor in I_i). Moreover, for any $1 \leq i \leq n$, we have $L_{J_i}(\mathsf{state}_i(\rho)) \geq L_{J_n}(\mathsf{state}_n(\rho))$, since all predecessors of J_i are in J_{i-1} by definition. We will show that $L_{J_{i_0}}(\mathsf{state}_{i_0}(\rho)) \leq L_{J_{i_0-1}}(\mathsf{state}_{i_0-1}(\rho)) - \epsilon^2/2$, which thus implies the desired inequality.

There exists $a \in J_{i_0-1}$ and $b \in I_{i_0} \setminus J_{i_0}$ with $(a,b) \in E$. Let us write $\lambda \mathbf{v} = \mathsf{state}_{i_0-1}(\rho)$. By Proposition 8.5.4, we have $\lambda_a \geq \epsilon$. Moreover, the transition from step $i_0 - 1$ to i_0 must be a delay since vertex a has two successors; a must have a successor in J_{i_0}, thus different than b, since $a \in J_{i_0-1}$. By Proposition 8.5.10, the edge (a,b) receives a probability of at least $\epsilon/2$ (in Lemma 8.5.1). The inequality follows. □

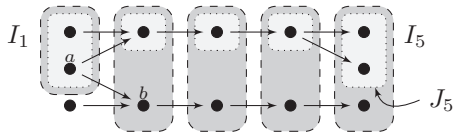


Figure 8.5: Sets I_i and J_i for the cycle of Fig. 3.1

Let $a \in J_{i_0-1}$ such that there exists $b \in I_{i_0} \setminus J_{i_0}$ with $(a,b) \in E(\gamma(\pi))$. Let us write $\lambda \mathbf{v} = \mathsf{state}_{i_0}(\rho)$. Then, $L_{J_{i_0}}(\mathsf{state}_{i_0}(\rho)) \leq L_{J_{i_0} \cup \{b\}}(\mathsf{state}_{i_0}(\rho)) - \epsilon$ since $\lambda_b \geq \epsilon$ by Proposition 8.5.4.

The previous proposition already gives an insight into the proof, since it follows that no non-forgetful cycle can be repeated infinitely often under strategy $\sigma_\delta^{\mathsf{P}}$. However, one also needs to show that switching between different cycles cannot lead to Controller's winning. Thus, the last tool we need before the proof is the following theorem, which will allow us to factor any paths into cycles with the same folded orbit graphs.

We will use Theorem 4.5.2 as in Chapter 4 as follows. Given a cycle π factorized as $\pi = \pi_1 \pi_2 \dots \pi_n$, in which all π_i are cycles from the same region-states, we consider a linearly ordered complete graph $G = (V, E)$ with $V = \{1, \dots, n\}$ and E is defined by the usual order on integers. The coloring function f is given as follows. For all $i < j$, we define $f((i,j)) = \Gamma(\pi_i \pi_{i+1} \dots \pi_j)$. Thus, a monochromatic path given by the theorem corresponds to a factorization into $\pi = \pi_0 \cdot (\pi_{\alpha_1} \dots \pi_{\alpha_2 - 1}) \cdot (\pi_{\alpha_2} \dots \pi_{\alpha_3 - 1}) \cdot \dots$ such that all $\Gamma(\pi_{\alpha_j} \dots \pi_{\alpha_{j+1}-1})$ define the same graph.

Note that Ramsey's theorem would actually suffice for our proof but it would give an exponentially larger bound than Theorem 4.5.2. Consequently, the proof below will bound by a simple exponential the number of steps to the "failure" under any strategy in non-robust systems.

Proof: No winning aperiodic lassos implies no robust safety. To get a contradiction, fix any winning strategy σ for Controller and let ρ be the infinite run $\mathsf{Outcome}_{\mathcal{A}}^{\delta}(\sigma, \sigma_\delta^{\mathsf{P}})$. Let π be the projection of ρ into regions. By definition of $\sigma_\delta^{\mathsf{P}}$, π is a non-punctual path. Let us write $\pi = \pi_0 \pi_1 \dots \pi_n \dots$ such that all π_i are accepting cycles from a same state. Now, $\Gamma(\cdot)$ associates each cycle π to its folded orbit graph. Since any such graph has $|\mathcal{C}| + 1$ nodes, and all its nodes are labelled with the same region, the codomain of Γ has size $r = 2^{(|\mathcal{C}|+1)^2} \times |\mathcal{R}(\mathcal{A})|$. Let $n = \lceil 2/\epsilon^2 \rceil + 1$ and N large enough so that $\lceil \sqrt{N/r - 2} \rceil - 1 \geq n 2^{(|\mathcal{C}|+1)^2}$. We extract π' the prefix of π with N factors, and apply Theorem 4.5.2 on π' where Γ is the coloring function. This gives a factorization $\pi' = \pi'_0 \pi'_1 \dots \pi'_{n'} \pi'_{n'+1}$, with $n' = n 2^{(|\mathcal{C}|+1)^2}$, where $\Gamma(\pi'_1) = \dots = \Gamma(\pi'_{n'})$, and π'_i are obtained by concatenating one or several consecutive π_i. By hypothesis, some power k of π'_i is non-forgetful, with $k \leq 2^{(|\mathcal{C}|+1)^2}$ since this is the number of folded orbit graphs for a fixed labelling function. But since the folded orbit graphs are the same for all π'_i, this power is the same for all factors. Moreover, for the same reason, $\Gamma(\pi_i'^k) = \Gamma(\pi'_i \pi'_{i+1} \dots \pi'_{i+k-1})$. Hence, we can factorize π' again into $\pi' = \pi'_0 \pi''_1 \pi''_2 \dots \pi''_n \pi''_{n+1}$, where $\Gamma(\pi''_1) = \dots = \Gamma(\pi''_n)$ and all are non-forgetful; while π'_0 and π''_{n+1} are arbitrary non-punctual paths. Moreover, π''_i, for $1 \leq i \leq n$, must be progress cycles too. In fact, otherwise there is some clock $x \in \mathcal{C}$ that is never reset along π'. But because $\sigma_\delta^{\mathsf{P}}$ ensures a

time progress of ϵ at each delay, this means that π' contains delays of at least $n\epsilon^2/2 > 1$, so we cannot have $\mathsf{first}(\pi'_1) = \mathsf{last}(\pi'_n)$ since the integer part of the clock x changes, and so does the region since all clocks are assumed to be bounded. Now, we have $I(\pi''_i) = I(\pi''_j)$ for any $1 \leq i, j \leq n$, so the functions $L_{I(\pi''_i)}$ are the same for all $1 \leq i \leq n$. If we write ρ_i the state reached in ρ following $\pi'_0\pi''_1 \dots \pi''_i$, then we get, by Lemma 8.5.15, $L_{I(\pi''_1)}(\rho_n) \leq L_{I(\pi''_1)}(\rho_0) - n\epsilon^2/2$. This is a contradiction since $0 \leq L_{I(\pi'_1)} \leq 1$ and $n\epsilon^2/2 > 1$.

8.6 Aperiodic Lassos Implies Robustness

We now prove that if $\mathcal{R}(\mathcal{A})$ contains an aperiodic non-punctual forgetful B-winning lasso, then there is a strategy for Controller in $\mathcal{G}^{\mathsf{cons}}_\delta(\mathcal{A})$ ensuring robust safety. In the proof, we use *shrinking* techniques from Chapters 5 and 7, and the topological semantics of [GHJ97]. The basic idea is the following. We prove that Controller can always ensure following one iteration of a (finite) non-punctual cycle. This ensures that the the cycle of an accepting lasso is reachable. Then, if the cycle is aperiodic, we show that Controller can also ensure to repeatedly move inside a set at the "middle" of the target region. This provides a winning strategy for Controller.

8.6.1 Controllable Predecessors

We will express "controllable predecessors" using shrunk DBMs without parameters from Chapter 3.

Consider an edge $e = (\ell, g, \sigma, R, \ell')$. For any set $Z \subseteq \mathbb{R}^{\mathcal{C}}_{\geq 0}$, we define the *controllable predecessors of* Z as follows:

$$\mathsf{CPre}^\delta_e(Z) = \mathsf{Pre}_{\geq\delta}(\mathsf{shrink}_{[-\delta,\delta]}(g \cap \mathsf{Unreset}_R(Z))).$$

Intuitively, $\mathsf{CPre}^\delta_e(Z)$ is the set of valuations from which Controller can ensure reaching Z in one step, following the edge e. In fact, it can delay in $\mathsf{shrink}_{[-\delta,\delta]}(g \cap \mathsf{Unreset}_R(Z))$ with a delay of at least than δ, where under any perturbation in $[-\delta, \delta]$, the valuation satisfies the guard, and it ends, after reset, in Z. We extend this operator to paths as expected. Note that CPre^0_e is the usual predecessor operator without perturbation: If $N = \mathsf{CPre}^0_\pi(M)$ for some sets N, M and path π, then, N is the set of all valuations that can reach some valuation in M following π.

It immediately follows from Lemma 3.4.5 that the controllable predecessors of shrunk DBMs are shrunk DBMs, which are computable.

Lemma 8.6.1. *Let $e = (\ell, g, \sigma, R, \ell')$ be an edge. Let M and N be non-empty DBMs such that $N = \mathsf{CPre}^0_e(M)$. Then, for any SM P, there exists an SM Q such that $(N', Q) = \mathsf{CPre}^\delta_e((M, P))$ for some $N \subseteq N'$ and $\mathsf{N} = \mathsf{N}'$.*

Notice that, the set (N', Q) given by the previous lemma could be empty. We show in the following lemma, that the controllable predecessors of open sets along non-punctual paths are non-empty for small enough $\delta > 0$.

Lemma 8.6.2. *Let π be a non-punctual path from region r to s. Let $s' \subseteq s$ such that there exists $\nu' \in s'$ and $\epsilon > 0$ with $\mathsf{Ball}_{d_\infty}(\nu', \epsilon) \cap s \subseteq s'$. Then, $\mathsf{CPre}^\delta_\pi(s')$ is non-empty for small enough $\delta > 0$.*

Proof. Note that the requirement on s' says that s' should have non-empty interior in the d_∞-topology induced on the set s. Consider such a set s' and a valuation $\nu' \in s'$ with $B = \mathsf{Ball}_{d_\infty}(\nu', \epsilon) \cap s \subseteq s'$ for some $\epsilon > 0$. Let ρ be any run along π from some $\nu \in r$ to ν', and let $u = (d_i, e_i)_{1 \leq i \leq n}$ denote the

timed trace of ρ. Assume that $d_i > 0$ for all $1 \leq i \leq n$. Choose $0 < \epsilon' < \epsilon$ so that $d_i \geq \epsilon'$ for all i, and all timed traces in $\mathsf{Ball}_d(u, \epsilon')$ are lead to some point in B. This is possible since all delays are positive, by Prop. 8.3.4 (π is non-punctual), and by the fact that the value of each clock at the last state of a run is continuously determined by the delays in the timed trace. Let $\delta \leq \epsilon'$. Now, starting at ν, if Controller plays according to u, then the timed trace of any outcome belongs to $\mathsf{Ball}_d(u, \epsilon')$, thus is feasible and ends in B. More precisely, Controller's strategy that ensures reaching B consists simply in playing u from ν. Therefore, $\nu \in \mathsf{CPre}_\pi^\delta(s')$.

If not all delays are positive, then one can choose another run ρ' with this property, and such that $d(u, \mathsf{ttrace}(\rho')) \leq \epsilon'$ since $\mathsf{ttrace}(\pi)$ is open, and use ρ' in the above proof. □

8.6.2 Winning Under Perturbations

Let $\pi_0\pi$ denote a non-punctual aperiodic lasso, where π is the cycle. By definition, for any $n \geq 1$, $\Gamma(\pi^n)$ is strongly connected. We first show that there exists $n \geq 1$ such that $\Gamma(\pi^n)$ is a complete graph.

Lemma 8.6.3. *Let π be an aperiodic cycle. Then, there exists $n \leq |\mathcal{C}_0| \cdot |\mathcal{C}_0|!$ such that $\Gamma(\pi^n)$ is a complete graph.*

Proof. For any vertex v, there exists $n_v \leq |V|$ such that $(v, v) \in \Gamma(\pi^{n_v})$ since $\Gamma(\pi)$ is strongly connected. Let $m = \mathsf{lcm}_{v \in V}(n_v)$. Then, $(v, v) \in \Gamma(\pi^m)$ for any vertex $v \in V$. Note that $m \leq |V|!$. Let us write $\pi' = \pi^m$. Consider the length $n_{v,w}$ of the shortest path from v to w in $\Gamma(\pi')$ for any $v, w \in V$, and let $m' = \max_{v,w \in V} n_{v,w}$, which is finite since π' is strongly connected. We have, $(v, w) \in \Gamma(\pi'^{m'})$. In fact, $(v, v) \in \Gamma(\pi'^{m'-n_{v,w}})$ since $(v, v) \in \Gamma(\pi')$ and $(v, w) \in \Gamma(\pi'^{n_{v,w}})$. We have, $m' \leq |V|$, so n can be chosen as $|V| \cdot |V|!$. □

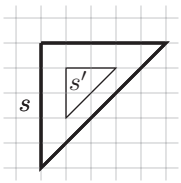


Figure 8.6: A subregion s' of s with smaller granularity

By the previous lemma, let us assume, in the rest of this section, that $\Gamma(\pi)$ is a complete graph (one can simply consider the lasso $\pi_0\pi^n$ for n large enough). Let s be a region with smaller granularity inside r, obtained so that the Hausdorff distance between r and s is positive. The choice of s is illustrated in Fig. 8.6. Note that s can be chosen so that it can be expressed by a DBM (with rational components). The construction is defined in the following lemma.

Lemma 8.6.4. *For any non-empty DBM M, there exists a non-empty DBM N such that $\mathsf{Ball}_{d_\infty}(\nu, \epsilon) \cap M \subseteq N$ for some $\nu \in M$, and for any shrinking matrix P with $(M, P) \neq \emptyset$, $N \subseteq (M, P)$.*

Proof. We define Q as follows. For any $x, y \in \mathcal{C}_0$, if $M_{x,y} = -M_{y,x}$, then $Q_{x,y} = Q_{y,x} = 0$. Otherwise, we have $\mathsf{M}_{x,y} \geq -\mathsf{M}_{x,y} + 1$, and in this case, we let $Q_{x,y} = Q_{y,x} = 1$. Let us show that $M - \epsilon Q \neq \emptyset$, for small enough $\epsilon > 0$. To show non-emptiness, it suffices to show that the constraints of the DBM $M - \epsilon Q$ do not imply a negative upper bound on zero [BY04]. For a

contradiction, assume there is a sequence of indices $x_1, x_2, \ldots, x_n \in \mathcal{C}_0$, with $x_1 = x_n$, such that $\sum_{i=0}^{n-1}(M_{x_i,x_{i+1}} - \epsilon Q_{x_i,x_{i+1}}) < (0, \leq)$ for all $\epsilon > 0$. This means that $\sum_{i=0}^{n-1} M_{x_i,x_{i+1}} \leq (0, \leq)$, but since M is non-empty, we have $\sum_{i=0}^{n-1} M_{x_i,x_{i+1}} = (0, \leq)$. But then, for all $1 \leq i \leq n-1$, we must have $\mathsf{M}_{x_i,x_{i+1}} = -\mathsf{M}_{x_{i+1},x_i}$, which implies $Q_{x_i,x_{i+1}} = 0$ by definition, thus we cannot have $\sum_{i=0}^{n-1}(M_{x_i,x_{i+1}} - \epsilon Q_{x_i,x_{i+1}}) < (0, \leq)$. To see that $\mathsf{M}_{x_i,x_{i+1}} = -\mathsf{M}_{x_{i+1},x_i}$, observe that if $-\mathsf{M}_{x_{i+1},x_i} < \mathsf{M}_{x_i,x_{i+1}}$ for some i, then by Lemma 3.3.3, for any $\alpha \in (0,1)$, there exists a valuation ν in M, and satisfying $\nu(x_i) - \nu(x_{i+1}) = \mathsf{M}_{x_i,x_{i+1}} - \alpha$. But since $\nu(x_j) - \nu(x_{j+1}) \leq \mathsf{M}_{x_j,x_{j+1}}$ for all $1 \leq j \leq n$, this implies that $\nu(x_1) - \nu(x_2) + \nu(x_2) - \nu(x_3) + \ldots - \nu(x_n) < \sum_{j=1}^{n-1} \mathsf{M}_{x_j,x_{j+1}} - \nu < 0$, which is a contradiction.

Now, we choose ϵ as above, and small enough so that $\mathsf{M}_{x,y} \neq -\mathsf{M}_{y,x}$ implies that $\mathsf{M}_{x,y} - \epsilon Q_{x,y} > -\mathsf{M}_{y,x} + \epsilon Q_{y,x}$. We let $N = M - \epsilon Q$. Then, given any shrunk DBM (M, P), we have $\mathsf{M}_{x,y} - \epsilon Q_{x,y} < \mathsf{M}_{x,y} - \delta P_{x,y}$, for small enough $\delta > 0$ as required. Since ϵ can be chosen arbitrarily small, N also has non-empty interior in the topology induced on M.

Note that N defined here could be given by scaling r, *i.e.* applying a homothety as in Figure 8.6. □

The following lemma states that along a cycle whose orbit graph is complete, the reachability relation between valuations is complete.

Lemma 8.6.5 ([BA11])**.** *Let π be a cycle of $\mathcal{R}(\mathcal{A})$ such that $\Gamma(\pi)$ is a complete graph. Then, for any pair $\nu, \nu' \in \mathsf{first}(\pi)$, there exists a run $\nu \xrightarrow{\pi} \nu'$.*

Proof: Winning aperiodic lassos implies robust safety. We write $r = \mathsf{first}(\pi)$. Let $s \subseteq r$ given by Lemma 8.6.4 applied to r. Because $\Gamma(\pi)$ is complete, we have, by Lemma 8.6.5, $r = \mathsf{CPre}^0_\pi(s)$. By Lemma 8.6.1, there exists a SM Q such that $(r, Q) = \mathsf{CPre}^\delta_\pi(s)$. By Lemma 8.6.2, (r, Q) is non-empty. By definition of s, for small enough $\delta > 0$, $s \subseteq (r, Q)$, so Controller has a strategy to always move inside s, at each iteration of π. Similarly, $\mathsf{CPre}^\delta_{\pi'}((r, Q))$ is also non-empty and therefore contains the initial state. Hence, Controller has a winning strategy. The strategy and a corresponding δ can be computed using Proposition 3.4.6, given an aperiodic lasso.

8.7 Algorithm

The algorithm for deciding robust Büchi acceptance follows from Lemma 8.4.1. It consists in looking for aperiodic non-punctual B-winning lassos in $\mathcal{R}(\mathcal{A})$. These lassos need not be simple, but we will bound their lengths, and give a polynomial space algorithm.

Lemma 8.7.1. *Let B be a Büchi objective in a timed automaton $\mathcal{A}$, and π be an aperiodic non-punctual B-winning cycle of $\mathcal{R}(\mathcal{A})$. Then, there exists a cycle π' with the same properties, with length at most $N = 2^{(|\mathcal{C}|+1)^2+1} \times |\mathcal{R}(\mathcal{A})|$.*

Proof. We label all prefixes $\pi_{1\ldots i}$ with $(\Gamma(\pi_{1\ldots i}), b_i)$, where $b_i = \top$ if $\pi_{1\ldots i}$ visits a location in B, and $\bot$ otherwise. The number of labels is at most $N = 2^{(|\mathcal{C}|+1)^2+1} \times |\mathcal{R}(\mathcal{A})|$. We argue that a cycle π' of length less than N can be extracted from π, if $|\pi| \geq N$. In fact, in this case, we must have $i < j$ such that $\Gamma(\pi_{1\ldots i}) = \Gamma(\pi_{1\ldots j})$ and $b_i = b_j$. But then, removing the infix $\pi_{i+1\ldots j}$ yields a shorter cycle, which visits B if $\pi_{1\ldots j}$ does so, and with the same folded orbit graph. By repeating this procedure, we obtain a path π' of length less than N, which is non-punctual aperiodic B-winning. □

The algorithm starts by guessing, using polynomial space, a state q_1 reachable by a non-punctual path in $\mathcal{R}(\mathcal{A})$. From q_1, one can guess an aperiodic non-punctual B-winning cycle in polynomial space as follows. We visit a path $q_1e_1q_2e_2\dots q_i$ nondeterministically, and only keeping in memory the graph $G_i = \Gamma(q_1e_1q_2e_2\dots q_i)$ and a boolean indicating whether a winning state has been visited. Thanks to the operator $\oplus$, for any transition $q_ie_iq_{i+1}$, one can compute G_{i+1} using only G_i and this transition. Whenever the state q_1 is visited again, the algorithm checks whether the current graph G_i is aperiodic, by Lemma 8.7.2. If so, it stops and accepts. Otherwise, it stops and rejects after N steps.

Lemma 8.7.2 ([Sta12]). *Given a cycle π, one can decide in polynomial space whether $\Gamma(\pi)$ is aperiodic.*

To actually compute a winning strategy, one can find a lasso as described above, while keeping the whole path $q_1e_1q_2\dots q_n$ in memory. Then, one can extract a subset as in Lemma 8.6.4, and apply Corollary 3.4.5 to the cycle and to the prefix. This computation only takes polynomial time in the size of the path, but since the lasso can have exponential size, the computation takes worst-case exponential time.

8.8 PSPACE-hardness of robust synthesis

We show PSPACE-hardness of the parametrised robust controller synthesis problem, both for reachability and safety objectives.

We reduce the halting problem for linearly-bounded deterministic Turing machines to our synthesis problem. Let $\mathcal{M}$ be a linearly-bounded deterministic Turing machine, and w_0 be an input to $\mathcal{M}$. Let N be the bound on the tape of $\mathcal{M}$ when simulating on input word w_0 (N is linear in $|w_0|$). We assume the alphabet is $\{a, b\}$, and we encode the content of the i-th cell C_i of $\mathcal{M}$ using a clock x_i with the following convention: when the module is entered, cell C_i contains an a whenever $x_i < 1$, and it contains a b whenever $x_i > 2$. To simulate a transition $(q, \alpha, q', \beta, \mathsf{dir})$ of $\mathcal{M}$, where $\alpha, \beta \in \{a, b\}$ and $\mathsf{dir} \in \{-1, 1\}$, we build a module as in Fig. 8.7 for every index i such that both i and $i + \mathsf{dir}$ lie between 1 and N. Along this module, after any initial delay of duration $u_0 \in (2, 3)$, cell C_j contains a iff $x_j < 4$, and it contains b iff $x_j > 4$. Transitions between p_j and p_{j+1}, for $j \neq i$, ensure preservation of this encoding. Between p_i and p_{i+1}, the module checks that $C_i = \alpha$ (through guard $g_{\alpha,i}$), and replace this content with β (through reset $Y_{\beta,i}$). This way, one run through the module updates the content of the tape and the position of the tape head according to the selected transition of $\mathcal{M}$.

We complete the construction with an initialization module (encoding input w_0 on the tape). We write $\mathcal{A}$ for this timed automaton and write halt for the halting location (which is made a sink).

We show the following lemma:

Lemma 8.8.1. *Let s_0 be the initial configuration of $\mathcal{A}$. The following three properties are equivalent:*

1. *$\mathcal{M}$ halts on w_0;*
2. *Controller has a robust strategy in $\mathcal{A}$ to reach halt, which can be perturbed by at most $\delta < \frac{1}{4N}$;*
3. *Controller has no robust strategy in $\mathcal{A}$ to avoid halt, whatever the value of $\delta > 0$.*

Proof. We first prove the equivalence of the two first properties.

First assume that $\mathcal{M}$ halts on w_0. We describe a robust winning strategy for the Controller in $\mathcal{A}$. We consider the simulation of the transition $(q, \alpha, q', \beta, \mathsf{dir})$ of $\mathcal{M}$. Assume state (q, i) is entered with

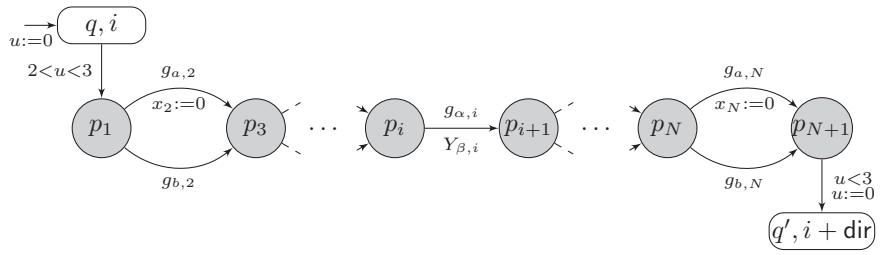


Figure 8.7: Simulation transition $(q, \alpha, q', \beta, \mathsf{dir})$ where $\alpha, \beta \in \{a, b\}$. Index i is such that $1 \leq i \leq N$ and $1 \leq i + \mathsf{dir} \leq N$. Guards and resets are defined as follows: $g_{a,j}$ is $(x_j < 4 \wedge u < 3)$ and $g_{b,j}$ is $(x_j > 4 \wedge u < 3)$, while $Y_{a,j}$ is $\{x_j\}$ and $Y_{b,j}$ is empty.

some valuation v_0 that properly encodes some content of the tape of $\mathcal{M}$: if cell C_j contains an a, then $v_0(x_j) < 1$, and if cell C_j contains a b, then $v_0(x_j) > 2$. We describe a strategy for Controller, and then pick a corresponding outcome (with perturbations). The candidate strategy is as follows: leave (q, i) when $u = 2.5$, and in each of the p_j's, wait for δ time units, and then takes the next feasible transition.

Under that strategy, and assuming that Perturbator perturbs by $-\delta \leq \epsilon_0 \leq \delta$ on transition leaving (q, i), and by $-\delta \leq \epsilon_j \leq \delta$ on transition leaving p_j, the valuation when arriving in p_j is v_j (resp. in $(q', i + \mathsf{dir})$ is v'_0) such that:

- if $i \neq j$ and cell C_j contains a b, or if $i = j$ and $\beta = b$, then for every $1 \leq k \leq N + 1$, $v_k(x_j) = v_0(x_j) + 2.5 + (k-1)\delta + \sum_{h=0}^{k-1} \epsilon_h$;
- if $i \neq j$ and cell C_j contains an a, or if $i = j$ and $\beta = a$, then for every $1 \leq k \leq j$, $v_k(x_j) = v_0(x_j) + 2.5 + (k-1)\delta + \sum_{h=0}^{k-1} \epsilon_h$, $v_{j+1}(x_j) = 0$, and for every $j + 2 \leq k \leq N + 1$, $v_k(x_j) = (k-j-1)\delta + \sum_{h=j+1}^{k-1} \epsilon_h$.

Note that in all cases, since $\delta < \frac{1}{4N}$, if cell C_j contains an a, then $v_j + \delta + \epsilon_j \models g_{a,j}$, and if cell C_j contains a b, then $v_j + \delta + \epsilon_j \models g_{b,j}$. Note also that if either $i \neq j$ and cell C_j contains an a, or if $i = j$ and $\beta = a$, then $v'_0(x_j) < 1$, and if either $i \neq j$ and cell C_j contains a b, or if $i = j$ and $\beta = b$, then $v'_0(x_j) > 2$. This means that whatever the perturbation, at the end of the module the clocks properly encode the new content of the tape of $\mathcal{M}$ after the mentioned transition has been taken.

We can apply this strategy in every module simulating a transition of the Turing machine. Since $\mathcal{M}$ halts on w_0, the halt state can be reached by following this strategy.

Conversely, if $\mathcal{M}$ does not halt on w_0, then when playing without perturbations, Controller cannot reach the halting state. In particular, Controller has no robust strategy.

Now we should realize that if the Turing machine halts on w_0, then every non-blocking (standard) execution in $\mathcal{A}$ leads to halt. So in particular, there is no robust strategy for avoiding halt. If the Turing machine does not halt, then the same strategy as above is robust and avoids halt. This implies the equivalence with the third property. □

In the above construction some clocks may not be bounded. However we can modify the construction to ensure that all clocks are always bounded. We assume that an a in cell C_j is encoded with clock constraint $x_j < 1$, and a b with clock constraint $2 < x_j < 4$ when entering the module.

Assuming this, the previous module ensures that, when leaving the module, either $x_j < 1$ (in case cell C_j now contains an a), or either $2 < x_j < 4$ or $4 < x_j < 7$ (the two possible cases for cell C_j to contain a b). It is easy to plug at the end of the module another module, which ensures that $2 < x_j < 4$ when cell C_j contains an a, and $x_j < 1$ when cell C_j contains a b. Then we can plug at the end of this second module another one which ensures that $x_j < 1$ when cell C_j contains an a, and $2 < x_j < 4$ when cell C_j contains a b, which is the initial encoding. Using this trick, the constructed timed automata has only bounded clocks.

8.9 Conclusion

We presented an algorithm for synthesizing robust strategies for Büchi objectives under the conservative perturbation game semantics, for an unknown perturbation parameter. Our results make a link between controller synthesis problems and recent work on convergence phenomena and entropy. In fact, if all locations are accepting, then a timed automaton is robustly controllable in our sense if, and only if it is *thick*, or equivalently, it has finite entropy, as defined in [BA11]. Technically, the distinction between forgetful and non-forgetful cycles was made, and their relation to characterization was already established in [BA11]. Our results show that under our perturbation game semantics, the "convergent runs" actually become blocking. This problem has lower complexity than the same problem for the excess perturbation game semantics, since even reachability is EXPTIME-complete (Chapter 7).

In order to fully treat the robust controller synthesis problem, one needs to consider timed games. We intend to investigate possible extensions of this work to timed games, starting from turn-based timed games, where Perturbator entirely determines the move in some locations. This could require generalizing the notion of aperiodicity from paths to trees since Controller can no more ensure to follow a given path.

Chapter 9

Weighted Timed Games

9.1 Introduction

Weighted timed automata are an extension of timed automata, introduced in [ATP01, BFH+01], which allow to express and solve optimization problems on timed systems. In this model, timed automata are enriched with a special cost variable whose value grows with a given constant derivative at each location. This variable can model, for instance, resource consumption, and one can find the cheapest run that allows to reach a given location. In fact, it was established in [BBBR07] that the optimal reachability problem for weighted timed automata is PSPACE-complete.

The model naturally extends to games. Some partial decidability results for weighted timed games have been proposed in [ABM04, BCFL04]. However, the problem is undecidable in the general case [BBR05b, BBM06] even for a fixed number of clocks.

Both undecidability results are based on encodings of the configurations of Minsky machines. In these works, simulating the computations of these machines require distinguishing clock valuations with arbitrary precision, and the ability of checking for equalities. This suggests that the problem might become decidable when one introduces perturbations in the semantics, so as to disallow too precise encodings.

The goal of this chapter is to investigate the possibility of obtaining decidability results on weighted timed games when these are subject to perturbations. Our results are unfortunately mainly negative: in both excess and conservative perturbation game semantics, cost optimal reachability remains undecidable on weighted timed games. On weighted timed automata, we show that the problem is PSPACE-complete in the conservative perturbation game semantics, and that it is undecidable in the excess perturbation game semantics.

9.2 Cost-Optimal Reachability

Let us first present the cost-optimal reachability problems in the exact setting.

Recall that we write $(\ell, \nu, c)|_{\mathsf{c}} = c$ and $(\ell, \nu, c, d, e)|_{\mathsf{c}} = c$ for any state (ℓ, ν, c) or (ℓ, ν, c, d, e) of a weighted timed game in the exact or perturbed semantics. All related definitions can be found in Chapter 2. Given a target location ℓ, we define the cost of a run in any semantics as follows.

$$\mathsf{cost}^{\ell}(\rho) = \inf\{\mathsf{state}_i(\rho)|_{\mathsf{c}} \mid 1 \leq i \leq |\rho|, \mathsf{loc}(\mathsf{state}_i(\rho)) = \ell\}.$$

Hence, if ℓ is never reached, then the cost is ∞. Otherwise it is the infimum of the costs observed at location ℓ.

We will use the terminology of the perturbation game semantics in weighted timed games: Player 1 is called Controller, and Player 2 is called Perturbator. Given a weighted timed game $\mathcal{A}$, and a target location ℓ, we denote by $S_C([\![\mathcal{A}]\!])$ the set of Controller's strategies, and by $S_P([\![\mathcal{A}]\!])$ the

set of Perturbator's strategies in $[\![\mathcal{A}]\!]$. For any pair of strategies $(\sigma, \sigma') \in S_C([\![\mathcal{A}]\!]) \times S_P([\![\mathcal{A}]\!])$, we define the cost of the outcome of (σ, σ') as

$$\mathsf{cost}^{\ell}_{\sigma,\sigma'}([\![\mathcal{A}]\!]) = \mathsf{cost}^{\ell}(\mathsf{Outcome}_{[\![\mathcal{A}]\!]}(\sigma, \sigma')).$$

We are interested in the least cost that Controller can ensure against any strategy of Perturbator. We define, for any weighted timed game $\mathcal{A}$ and target location ℓ,

$$\mathsf{value}(\mathcal{A}, \ell) = \inf_{\sigma \in S_C([\![\mathcal{A}]\!])} \sup_{\sigma' \in S_P([\![\mathcal{A}]\!])} \mathsf{cost}^{\ell}_{\sigma,\sigma'}([\![\mathcal{A}]\!]).$$

When $\mathcal{A}$ is a weighted timed automaton, we define $\mathsf{value}(\mathcal{A}, \ell)$ by interpreting it as a weighted timed game where Perturbator has no edges. Equivalently,

$$\mathsf{value}(\mathcal{A}, \ell) = \inf_{\rho \in \mathsf{Runs}([\![\mathcal{A}]\!])} \mathsf{cost}^{\ell}(\rho).$$

We are interested in the following decision problems.

Definition 9.2.1. *The* OptReach strategy upper bound problem *asks, given a weighted timed game $\mathcal{A}$, a target location ℓ, and a rational λ, whether there exists a strategy $\sigma \in S_C([\![\mathcal{A}]\!])$ such that*

$$\sup_{\sigma' \in S_P([\![\mathcal{A}]\!])} \mathit{cost}^{\ell}_{\sigma,\sigma'}([\![\mathcal{A}]\!]) \leq \lambda.$$

Thus the above problem asks whether one can find a strategy that guarantees an upper bound on the cost. In the following problem, we are interested in deciding an upper bound on the infimum of the cost that can be guaranteed by Controller.

Definition 9.2.2. *The* OptReach value upper bound problem *asks, given a weighted timed game $\mathcal{A}$, a target location ℓ, and a rational λ, whether $\mathit{value}(\mathcal{A}, \ell) \leq \lambda$.*

Clearly, a bound on the cost of some strategy implies the same bound on the value; the converse is not true.

This problem was also considered on weighted timed automata, that is, when Player 2 has no edges. In this case, the above problems simplify: The value upper bound problem consists in deciding an upper bound on $\inf_\rho(\mathsf{cost}^{\ell}(\rho))$, where ρ ranges over all runs of the given weighted timed automaton; while the strategy upper bound asks whether the bound is satisfied for some run.

Theorem 9.2.3 ([ATP01, BFH+01, BBBR07])**.** *The* OptReach *value and strategy upper bound problems are* PSPACE*-complete for weighted timed automata.*

Unfortunately, the OptReach strategy upper bound problem becomes undecidable on weighted timed games, even for a fixed number of clocks.

Theorem 9.2.4 ([BBR05b, BBM06])**.** *The* OptReach *strategy upper bound problem is undecidable for weighted timed games.*

The decidability of the OptReach *value* upper bound problem for weighted timed games is open.

9.3 Robust Cost-Optimal Reachability

We define variants of the OptReach problems under perturbations. We are again interested in small values of the perturbation parameters. In fact, we will consider the limit of the cost when the magnitude of the perturbations goes to 0.

Given a weighted timed game $\mathcal{A}$, and $\delta > 0$, let us denote by $S_P(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A}))$ (resp. $S_P(\mathcal{G}_\delta^{\mathsf{cons}}(\mathcal{A}))$) the set of strategies of Perturbator in $\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})$ (resp. $\mathcal{G}_\delta^{\mathsf{cons}}(\mathcal{A})$). We also define $S_C(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A}))$ (resp. $S_C(\mathcal{G}_\delta^{\mathsf{cons}}(\mathcal{A}))$) as the set of Controller's strategies in these semantics. Notice that $S_C(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A}))$ does not depend on δ, since Controller only has to suggest moves that satisfy the guards (a *winning* strategy will depend on δ though). In contrast, $S_C(\mathcal{G}_\delta^{\mathrm{cons}}(\mathcal{A}))$ does depend on δ since Controller is required to satisfy the guards even after perturbations. It is easy to see that $S_C(\mathcal{G}_\delta^{\mathrm{cons}}(\mathcal{A})) \subseteq S_C(\mathcal{G}_{\delta'}^{\mathrm{cons}}(\mathcal{A}))$ for any $\delta' < \delta$.

Given $\delta > 0$, a pair of strategies $(\sigma, \sigma') \in S_C(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})) \times S_P(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A}))$, and location ℓ, we define

$$\mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})) = \mathsf{cost}^\ell(\mathsf{Outcome}_{\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})}(\sigma, \sigma')).$$

We define similarly $\mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}_\delta^{\mathsf{cons}}(\mathcal{A}))$, for the conservative perturbation game semantics. Given a strategy $\sigma \in S_C(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A}))$, we define the *limit cost of* σ as follows.

$$\mathsf{lim\text{-}cost}^{\mathrm{exs}}_\sigma(\mathcal{A}, \ell) = \lim_{\delta \to 0} \sup_{\sigma' \in S_P(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A}))} \mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A})).$$

The limit is well defined since strategy σ is valid for any $\delta > 0$. Similarly, for $\sigma \in \mathcal{G}_{\delta_0}^{\mathsf{cons}}$, we let

$$\mathsf{lim\text{-}cost}^{\mathrm{cons}}_\sigma(\mathcal{A}, \ell) = \lim_{\substack{\delta \to 0 \\ 0 < \delta < \delta_0}} \sup_{\sigma' \in S_P(\mathcal{G}_\delta^{\mathsf{cons}}(\mathcal{A}))} \mathsf{cost}^\ell_{\sigma,\sigma'}(\mathcal{G}_\delta^{\mathsf{cons}}(\mathcal{A})).$$

Here, we take the limit for $0 < \delta < \delta_0$, such that $\sigma \in S_P(\mathcal{G}_{\delta_0}^{\mathsf{cons}}(\mathcal{A}))$ so that the strategy is valid. Thus, the limit-cost of a Controller's strategy σ is the cost it guarantees in the limit, against any strategy of Perturbator, when δ goes to 0.

We are interested in deciding whether Controller has a strategy that guarantees an upper bound on the limit-cost for a reachability objective.

Definition 9.3.1 (Lim-OptReach)**.** *The* Lim-OptReach strategy upper bound problem for the excess perturbation game semantics *asks, given a weighted timed game $\mathcal{A}$, a target location ℓ, and a rational λ, whether there exists a strategy $\sigma \in S_C(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A}))$ such that $\mathsf{lim\text{-}cost}^{exs}_\sigma(\mathcal{A}, \ell) \leq \lambda$.*

The Lim-OptReach strategy strict upper bound problem for the excess perturbation game semantics *asks, given the same input, whether there exists a strategy $\sigma \in S_C(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A}))$ such that $\mathsf{lim\text{-}cost}^{exs}_\sigma(\mathcal{A}, \ell) < \lambda$.*

We define similarly the Lim-OptReach strategy (strict) upper bound problem for the conservative perturbation game semantics.

We define the *limit value* as the infimum of the limit cost that can be guaranteed by Controller:

$$\mathsf{lim\text{-}value}^{\mathrm{exs}}(\mathcal{A}, \ell) = \inf_{\sigma \in S_C(\mathcal{G}_\delta^{\mathsf{exs}}(\mathcal{A}))} \mathsf{lim\text{-}cost}^{\mathrm{exs}}_\sigma(\mathcal{A}, \ell),$$

and

$$\mathsf{lim\text{-}value}^{\mathrm{cons}}(\mathcal{A}, \ell) = \inf_{\sigma \in \bigcup_{\delta>0} S_C(\mathcal{G}_\delta^{\mathsf{cons}}(\mathcal{A}))} \mathsf{lim\text{-}cost}^{\mathrm{cons}}_\sigma(\mathcal{A}, \ell).$$

We also consider deciding upper bounds on values:

Definition 9.3.2. *The* Lim-OptReach value upper bound problem for the excess (resp. conservative) perturbation game semantics *asks, given a weighted timed automaton $\mathcal{A}$, a target location ℓ, and a rational λ, whether $\mathsf{lim\text{-}value}^{exs}(\mathcal{A},\ell) \leq \lambda$ (resp. $\mathsf{lim\text{-}value}^{cons}(\mathcal{A},\ell) \leq \lambda$).*

We will also consider the restriction of these problems to weighted timed *automata*.

Notice that the strategy strict upper bound problem is equivalent to deciding whether the strict upper bound holds for the *value*, that is the infimum of the limit cost over all possible strategies.

We show that neither the excess perturbation nor the conservative perturbation game semantics renders the Lim-OptReach problems decidable on timed games. Moreover, the problem even becomes undecidable in weighted timed automata under the excess perturbation game semantics.

Theorem 9.3.3. *The Lim-OptReach strategy upper bound problem is undecidable for weighted timed automata under the excess perturbation game semantics, for a fixed number of clocks.*

Theorem 9.3.3 is a rather surprising result. It reveals that adding perturbations can render problems intractable, which is the opposite of a common belief [AB01, Frä99]. In this case, optimal reachability is PSPACE-complete for weighted timed automata under the exact semantics, but becomes undecidable under the excess perturbation game semantics.

The conservative robust semantics is more restrictive; we have already seen that reachability in timed automata is less costly than in the excess perturbation game semantics. In fact, we show that the Lim-OptReach value upper bound is PSPACE-complete:

Theorem 9.3.4. *The Lim-OptReach value upper bound problem is* PSPACE*-complete on weighted timed automata under the conservative perturbation game semantics.*

However the Lim-OptReach strategy upper bound problem is also undecidable in the conservative semantics on weighted timed games.

Theorem 9.3.5. *The Lim-OptReach strategy strict upper bound problem is undecidable for weighted timed games under the conservative perturbation game semantics, for a fixed number of clocks.*

The undecidability also holds in both semantics when we consider the problem for a fixed δ:

Theorem 9.3.6. *The following problem is undecidable: For any fixed $0 \leq \delta \leq \frac{1}{3}$, given a weighted timed game $\mathcal{A}$, a target location ℓ, and rational λ, decide whether*

$$\inf_{\sigma \in S_C(G)} \sup_{\sigma' \in S_P(G)} \mathsf{cost}^{\ell}_{\sigma,\sigma'}(G) < \lambda,$$

where G denotes either $\mathcal{G}^{\mathsf{exs}}_{\delta}(\mathcal{A})$ or $\mathcal{G}^{\mathsf{cons}}_{\delta}(\mathcal{A})$.

In the rest of this chapter, we present the proofs of the above theorems. The decidability result is obtained by adapting the corner-point abstraction [BBBR07], while the undecidability results are based on reductions from the halting problem for Minsky machines [Min67], and on the encoding of [BBM06].

9.4 Encoding Minsky Machines

Consider a Minsky machine with counters c_1 and c_2, and a list of instructions $I_1, \ldots, I_n$. Here, each instruction I_i, for $1 \leq i \leq n-1$, is an *incrementation for c_b* as,

$$c_b = c_b + 1\texttt{; goto } I_j,$$

for $b = 1$ or 2, or a *decrementation with zero-test for* c_b as,

$$\texttt{if } (c_b = 0) \texttt{ goto } I_j \texttt{ else } c_i = c_i - 1\texttt{; goto } I_{j'}.$$

The instruction I_n is the *ending instruction*, that is, the final state. The halting problem asks whether the instruction I_n is reachable, starting from the configuration $c_1 = 0$, $c_2 = 0$, at instruction I_1.

The proofs are based on the simulation of Minsky machines and the encoding of the instructions are based on that of [BBM06]. However all constructions must be modified in order to deal with perturbations. Our reduction uses 10 clocks $x, x', y, y', u, u', t, t', z, z'$. A counter of a Minsky machine with value n will be encoded by a pair of clocks x, x' with values $k_x + \frac{1}{2^n}$ and $k_{x'} + \frac{1}{2^n}$ for some integers $k_x, k_{x'}$. Here, k_x is called the *shift* of x. If α denotes the clock x', we let $\alpha' = x$, and $\alpha'' = x'$, and similarly for other clocks. A configuration of a Minsky machine with counter values $n, m \geq 0$, is entirely encoded by four clocks:

$$\begin{array}{ll} x = k_x + \frac{1}{2^n}, & x' = k_{x'} + \frac{1}{2^n}, \\ y = k_y + \frac{1}{2^m}, & y' = k_{y'} + \frac{1}{2^m}, \end{array} \tag{9.1}$$

for some shifts $k_x, k_{x'}, k_y, k_{y'}$. The redundancy in this encoding is necessary to cope with perturbations; this will be clear in the next section. We denote by $\mathbf{k}$ the vector of shifts, for all clocks, and by $\mathsf{code}_{\mathbf{k}}(n, m)$ the set of valuations satisfying (9.1). We also define $\mathsf{code}^{\epsilon}_{\mathbf{k}}(n, m)$ the set of valuations ν such that $\nu + \eta \in \mathsf{code}_{\mathbf{k}}(n, m)$, where $|\eta(\alpha)| \leq \epsilon$, $\eta(\alpha) = \eta(\alpha')$ for all clocks α, and moreover $\eta(x) = \eta(x') = 0$ whenever $n = 0$, and $\eta(y) = \eta(y') = 0$ whenever $m = 0$. In other terms, $\mathsf{code}^{\epsilon}_{\mathbf{k}}(n, m)$ is the set of valuations that encode a configuration with an error bounded by ϵ, except that the encoding of the counter value 0 is exact. Given shifts $\mathbf{k}$ and a valuation $\nu \in \mathsf{code}^{\epsilon}_{\mathbf{k}}(n, m)$, we denote $\mathsf{frac}(\nu) = \nu - \mathbf{k}$. This gives the fractional part of the clocks, except when $n = 0$, in which case the components x and x' are equal to 1. We say that a valuation ν encodes a configuration (n, m) of the machine if it satisfies (9.1) for some shift vector $\mathbf{k}$.

9.5 Lim-OptReach Under Excess Perturbation

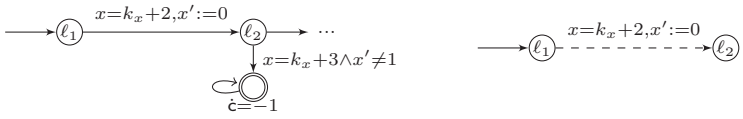
In this section, we reduce the halting problem on Minsky machines to the OptReach strategy upper bound problem on a weighted timed automaton we construct, which proves Theorem 9.3.3.

We define modules for incrementation and decrementation with zero-test instructions, which will, once combined, yield the reduction. The modules will be defined on a given list of clocks. For instance, if we describe a module $M(x, y, z, u, t)$ which uses the clocks x, y, z, u, t in its definition, then $M(z, y, x, t, u)$ is obtained simply by exchanging x and z, and u and t.

In this section, strategies and outcomes refer to the excess perturbation game semantics.

Unperturbed edges. Let us first present a construction that prevents Perturbator from perturbing the delays along an edge. The construction only applies to deterministic transitions (with equality constraints) and requires resetting one of the two clocks used in the encoding of a counter. Consider the timed automaton in Fig. 9.1(a).

At ℓ_2, Controller can go to the accepting state where the cost decreases to $-\infty$ if, and only if Perturbator has perturbed by a nonzero amount the transition from ℓ_1 to ℓ_2. Thus, Perturbator does not have interest in perturbing since its objective is to maximize the cost. If there has been no perturbation, the clock values are only increased or decreased by some integers. More precisely, the shifts of all clocks but x' increase by 2, and the shift of x' becomes 0. In the rest, we will use this



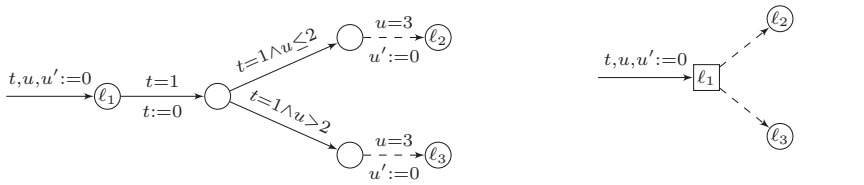
(a) An unperturbed edge.

(b) A simpler representation of an unperturbed edge.

Figure 9.1: Unperturbed edges. Perturbator has interest in not perturbing these transitions, since, in presence of any perturbation, Controller immediately can go to the target location and win with cost $-\infty$.

trick extensively to construct our modules. For better readability, we will represent such *unperturbed edges* by dashed arrows; when clear from context, we may omit the edges leading to accepting sink states (see Fig. 9.1(b)).

Ask-Perturbator module. In weighted timed automata, unlike in weighted timed games, Perturbator cannot suggest moves since it has no edges. However, a special construction allows letting Perturbator decide the successor location. We describe in Fig. 9.2(a) such a construction, which also ensures that the configuration is preserved, upto shifts.



(a) Letting Perturbator decide.

(b) Simpler representation of the automaton on the left.

Figure 9.2: Module that lets Perturbator decide a successor among ℓ_2 and ℓ_3.

The first edge is deterministic, and Perturbator *can* add any perturbation. Controller then distinguishes between positive and negative perturbations, and only has one possible move accordingly. We disallow perturbations (using unperturbed edges) at the edges leading to ℓ_2 or ℓ_3, so that the configuration is preserved upto shifts. More precisely, the shifts of all clocks x, x', y, y' increase by 3. To simplify the presentation of more complex modules, we will represent this module more compactly as in Fig. 9.2(b).

Reduction module. In the above modules, we have seen that configurations are preserved upto shifts, but the shifts could grow. We present now a module that reduces the shifts of all clocks. The module $\mathsf{Reduce}_{\mathbf{k}}(x, y, u, t)$ is constructed for each shift vector $\mathbf{k}$ (there will be a finite number of these), is deterministic, and contructed using unperturbed edges. Figure 9.3 defines one half of the module (named $\mathsf{PReduce}_{\mathbf{k}}(x, y, u, t)$). Any run from ℓ_1 to ℓ_2 leaves the encoded configuration unchanged, but only modifies the shifts. This automaton reduces the shift of the clocks x, y (to 3

and 2, respectively) but the shifts of x', y' increase by 5. The whole module $\mathsf{Reduce}_{\mathbf{k}}(x, y, u, t)$ is defined by concatenating $\mathsf{PReduce}_{\mathbf{k}}(x, y, u, t)$ and $\mathsf{PReduce}_{\mathbf{k}}(x', y', t, u)$, by merging the location ℓ_2 of the former with the location ℓ_1 of the latter. Note that u and t switch roles at the end of each $\mathsf{PReduce}_{\mathbf{k}}(\cdot)$; in fact, u, u' are still needed at location ℓ_2 to verify that there has been no perturbation, so they cannot both be reset. Module $\mathsf{PReduce}_{\mathbf{k}}$ satisfies the following property, which implies Lemma 9.5.2.

Lemma 9.5.1. *Let $\delta < \frac{1}{2}$. Assume $\mathsf{PReduce}_{\mathbf{k}}(x, y, u, t)$ is entered with valuation $\nu \in \mathsf{code}^{\epsilon}_{\mathbf{k}}(n, m)$ for some $\epsilon < \frac{1}{2}$. Controller has a strategy to either go to the target location with cost $-\infty$ or to reach location ℓ_2 with valuation ν' satisfying $\nu' = \mathsf{frac}(\nu) + \mathbf{k}'$ where $\mathbf{k}'$ is defined as follows: $k'_x = 2, k'_{x'} = k_{x'} + 2, k'_y = 1, k'_y = k_{y'} + 5$.*

In the above lemma, the bound on δ and ϵ ensure that valuations in $\mathsf{code}^{\epsilon}_{\mathbf{k}}(n, m)$, when perturbed by δ differ by less than an integer from the exact valuation $\mathsf{code}_{\mathbf{k}}(n, m)$.

Lemma 9.5.2. *Let $\delta < \frac{1}{2}$. Assume $\mathsf{Reduce}_{\mathbf{k}}(x, y, u, t)$ is entered with valuation $\nu \in \mathsf{code}^{\epsilon}_{\mathbf{k}}(n, m)$ for some $\epsilon < \frac{1}{2}$. Controller has a strategy to either go to the target location with cost $-\infty$ or to reach location ℓ_2 with valuation ν' satisfying $\nu' = \mathsf{frac}(\nu) + \mathbf{k}'$ where $\mathbf{k}'$ is defined as follows: $k'_x = 6, k'_{x'} = 2, k'_y = 5, k'_{y'} = 1$.*

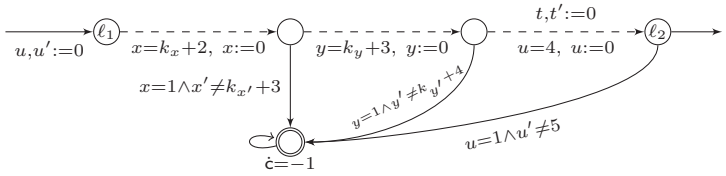


Figure 9.3: Module $\mathsf{PReduce}_{\mathbf{k}}(x, y, u, t)$, reducing the shifts of x and y.

Test Module. In order to verify the incrementation and decrementation, we will use the cost variable. We first show how one can add $1 + \mathsf{frac}(x)$ and $2 - \mathsf{frac}(x)$ to the cost variable, without changing the configuration. The construction is similar to [BBM06]; we adapt it using unperturbed edges.

The module $\mathsf{Add}^{1+x}_{\mathbf{k}}(x, u, t)$ depicted in Fig. 9.4 adds $1 + \mathsf{frac}(x)$ to the cost, leaving the configuration unchanged (upto shifts).

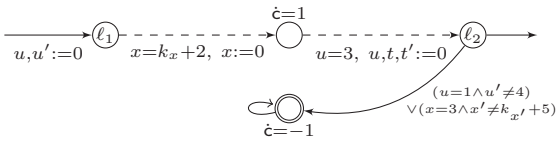


Figure 9.4: Module $\mathsf{Add}^{1+x}_{\mathbf{k}}(x, u, t)$.

Lemma 9.5.3. *Let $\delta < \frac{1}{2}$. Assume module $\mathsf{Add}^{1+x}_{\mathbf{k}}(x, u, t)$ is entered with valuation $\nu \in \mathsf{code}^{\epsilon}_{\mathbf{k}}(n, m)$ for some $\epsilon < \frac{1}{2}$. Controller has a strategy that ensures either reaching a target location with cost*

$-\infty$ or location ℓ_2 with valuation ν' satisfying $\nu' = \mathit{frac}(\nu) + \mathbf{k}'$ where $k'_x = 1$, and $k'_\alpha = k_\alpha + 3$ for all $\alpha \in \{x', y, y', z, z'\}$, while the cost increases by $1 + \mathit{frac}(x)$.

We define similarly a module $\mathsf{Add}_{\mathbf{k}}^{2-x}(x,u,t)$ that adds $2 - \mathsf{frac}(x)$ to the cost variable. The module is defined as in Fig. 9.4 except that c only increases (with slope 1) at location ℓ_1.

Lemma 9.5.4. *Let $\delta < \frac{1}{2}$. Assume module $\mathsf{Add}_{\mathbf{k}}^{2-x}(x,u,t)$ is entered with valuation $\nu \in \mathsf{code}_{\mathbf{k}}^{\epsilon}(n,m)$ for some $\epsilon < \frac{1}{2}$. Controller has a strategy that ensures either reaching a target location with cost $-\infty$ or location ℓ_2 with valuation ν' satisfying $\nu' = \mathit{frac}(\nu) + \mathbf{k}'$ where $k'_x = 1$, and $k'_\alpha = k_\alpha + 3$ for all $\alpha \in \{x', y, y', z, z'\}$, while the cost increases by $2 - \mathit{frac}(x)$.*

Now, concatenating modules $\mathsf{Add}_{\mathbf{k}}^{1+x}(x,u,t)$, $\mathsf{Add}_{\mathbf{k}'}^{2-x}(z,t,u)$, and $\mathsf{Add}_{\mathbf{k}''}^{2-x}(z,u,t)$, we obtain the module $\mathsf{Add}_{\mathbf{k}}^{5+x-2z}(x,z,u,t)$ that adds $5+\mathsf{frac}(x)-2\mathsf{frac}(z)$ to the cost and yields to a target location (we make the last location accepting). Similarly, by concatenation, we define $\mathsf{Add}_{\mathbf{k}}^{4+2z-x}(x,z,u,t)$ that adds $4+2\mathsf{frac}(z)-\mathsf{frac}(x)$ to the cost. One can append at the end of the module $\mathsf{Add}_{\mathbf{k}}^{4+2z-x}(x,z,u,t)$ an edge that increments the cost by 1, to define $\mathsf{Add}_{\mathbf{k}}^{5+2z-x}(x,z,u,t)$ (see Fig. 9.5). Finally, the module $\mathsf{Test}^{2z=x}(x,z,u,t)$ is defined by letting Perturbator choose whether to go to $\mathsf{Add}_{\mathbf{k}'}^{5+2z-x}(x,z,t,u)$ or to $\mathsf{Add}_{\mathbf{k}'}^{5+x-2z}(x,z,t,u)$ (here the new shift vector $\mathbf{k}'$ is due to the shift added by the ask-Perturbator module). Note that in both cases, the run ends in a target location. We have the following property.

Lemma 9.5.5. *Let $\delta < \frac{1}{2}$. If module $\mathsf{Test}^{2z=x}(x,z,u,t)$ is entered with valuation $\nu \in \mathsf{code}_{\mathbf{k}}^{\epsilon}(n,m)$ for some $\epsilon < \frac{1}{2}$, Controller has a strategy to ensure reaching a target location with cost at most $5 + |2\mathit{frac}(z) - \mathit{frac}(x)|$.*

Proof. Follows from Lemmas 9.5.4 and 9.5.3. □

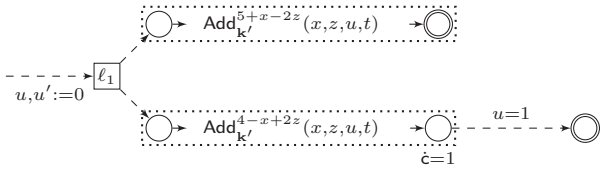


Figure 9.5: Module $\mathsf{Test}^{2z=x}(x,z,u,t)$ obtained by letting Perturbator choose between $\mathsf{Add}_{\mathbf{k}'}^{5+x-2z}(x,z,u,t)$ and $\mathsf{Add}_{\mathbf{k}'}^{5+2z-x}(x,z,u,t)$. The latter module is defined by appending one unperturbed transition with forcing the game to spend exactly one time unit at the last location of $\mathsf{Add}_{\mathbf{k}'}^{4+2z-x}(x,z,u,t)$.

Incrementation Module $\mathsf{Aut}_{\mathbf{k}}{}^{c_1,+}$. We now define a module $\mathsf{Aut}_{\mathbf{k}}{}^{c_1,+}$, given in Fig. 9.6, that simulates the incrementation of counter c_1. For the sake of discussion, let us assume first that there is no perturbation. When the module is entered with a valuation in $\mathsf{code}_{\mathbf{k}}(n,m)$, we expect Controller to choose the delays so that $z = 1 + \frac{\mathsf{frac}(x)}{2}$ at location D. From this point on, the clocks z, z' will switch roles with x, x'. Thus, this corresponds to incrementing the counter c_1 by 1. At location D, Perturbator can either decide to "test" the incrementation has been correctly performed by going to the test module, or to continue the simulation by first passing through the reduction module. Here, $\mathsf{Instr}_{\mathbf{k}}{}^{j}$ refers to a module among $\mathsf{Aut}_{\mathbf{k}}{}^{c_b,+}$, $\mathsf{Aut}_{\mathbf{k}}{}^{c_b,-}$ for $b \in \{1,2\}$ (to be defined

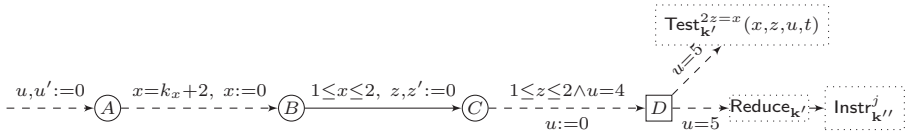


Figure 9.6: Module $\mathsf{Aut}_{\mathbf{k}}{}^{c_1,+}(x,y,z,u,t)$. The module $\mathsf{Reduce}_{\mathbf{k}'}$ refers to $\mathsf{Reduce}_{\mathbf{k}'}(z,y,u,t)$, and the module $\mathsf{Instr}^j_{\mathbf{k}''}$ to $\mathsf{Instr}^j_{\mathbf{k}''}(z,y,x,u,t)$.

next) according to the instruction I_j. In an actual run, Perturbator can perturb the value of z chosen by Controller, but only by δ. So at D, if Perturbator goes to the test module the cost is at most $5+O(\delta)$, provided that Controller has played correctly. Otherwise, the simulation carries on with $z \in \frac{\mathsf{frac}(x)}{2} \pm \delta$. The following lemma states this formally.

Lemma 9.5.6. *Let $\delta < \frac{1}{2}$. Assume module $\mathsf{Aut}_{\mathbf{k}}{}^{c_1,+}$ is entered with valuation $\nu \in \mathsf{code}^{\epsilon}_{\mathbf{k}}(n,m)$ for some $\epsilon > 0$ with $\delta + \epsilon/2 < \frac{1}{2}$, and cost 0. Then, Controller has a strategy that ensures that either module $\mathsf{Instr}_{\mathbf{k}}{}^j$ is entered with a valuation $\nu' \in \mathsf{code}^{\delta+\epsilon/2}_{\mathbf{k}'}(n+1,m)$ for some $\mathbf{k}'$, or the target location is reached with cost at most $5+2\delta$.*

Proof. We let Controller go to the target location after unperturbed edges, whenever there has been a nonzero perturbation. So, to describe Controller's strategy, it suffices to define its delay in location B since the other edges are deterministic. We let Controller delay $2 - \frac{\mathsf{frac}(\nu(x))}{2}$ in B. This delay can be perturbed by δ. Then, the delay at location C must be $1 + \frac{\mathsf{frac}((\nu(x))}{2} \pm \delta$. Upon arrival to D, Perturbator either chooses to go to the module $\mathsf{Reduce}_{\mathbf{k}'}$ and the play continues at $\mathsf{Instr}^j_{\mathbf{k}''}$, or it chooses to the test module. By Lemma 9.5.5, the outcome is then at most $5+2\delta$.

Now, if the game proceeds to $\mathsf{Instr}_{\mathbf{k}}{}^j$, then the encoding for counter m has not changed thanks to the unperturbed edges. For counter n, we have initially $\mathsf{frac}(x) = \frac{1}{2^n} + \eta$ with $\eta \in [-\epsilon, \epsilon]$. We get that upon arrival to D, we must have $\mathsf{frac}(z) = \frac{1}{2^{n+1}} + \eta/2 \pm \delta$. Hence $\nu' \in \mathsf{code}^{\delta+\epsilon/2}_{\mathbf{k}'}(n+1,m)$, where ν' denotes the valuation upon arrival to D. □

Decrementation Module $\mathsf{Aut}_{\mathbf{k}}{}^{c_1,-}$. We now define a module that simulates the decrementation with zero-test instruction. We explain the expected behaviour of the module. The zero-test is done at the first location A: the counter c_1 equals 0 if exactly one time unit after the arrival at A, we have $x = k_x + 2$. Recall that the encoding is exact for counters having the value 0. Otherwise, the run moves to either B or C'. In the former, Controller is to wait for $1 - 2\mathsf{frac}(x)$ time units, so that $z = 1 + 2\mathsf{frac}(x)$ upon arrival to D. This simulates a decrementation of c_1, upto an error of δ, since the edge from B to C can be perturbed. If the decremented value of the counter c_1 is 0, then the new value for z can be chosen as exactly 2 by Controller along the path through C'.

Lemma 9.5.7. *Assume module $\mathsf{Aut}_{\mathbf{k}}{}^{c_1,-}$ is entered with valuation $\nu \in \mathsf{code}^{\epsilon}_{\mathbf{k}}(n,m)$ and cost 0. Then,*

- *If $n = 0$, Controller has a strategy to ensure reaching either $\mathsf{Instr}_{\mathbf{k}}{}^j$ with the same configuration upto shifts and cost, or an accepting location with cost 0.*
- *If $n \ge 1$, and $\epsilon < \frac{1}{2}$, the play cannot reach $\mathsf{Instr}_{\mathbf{k}}{}^j$.*

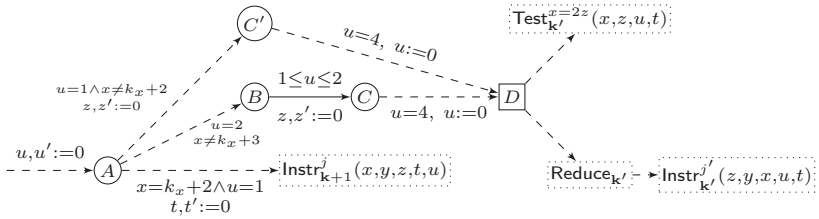


Figure 9.7: Module $\mathsf{Aut}_{\mathbf{k}}{}^{c_i,-}(x,y,z,u,t)$ with zero test. The module $\mathsf{Reduce}_{\mathbf{k}'}$ refers to $\mathsf{Reduce}_{\mathbf{k}'}(z,y,u,t)$.

- *If $n \geq 1$, Controller has a strategy that ensures that either module $\mathsf{Instr}_{\mathbf{k}}{}^{j'}$ is reached with a valuation $\nu' \in \mathsf{code}^{\delta+2\epsilon}_{\mathbf{k}'}(n-1,m)$ for some $\mathbf{k}'$, or the target location is reached with cost at most $5+\epsilon+2\delta$. Moreover, if $n=1$, then Controller can ensure that $\nu'(x) = k'_x + 1$.*

Proof. If $n = 0$, then $\nu(x) = k_x + 1$. In this case, Controller can choose to move, after a delay of 1, to $\mathsf{Instr}_{\mathbf{k}}{}^{j}$. Controller wins by moving to a target location if Perturbator perturbs, otherwise the play continues in $\mathsf{Instr}_{\mathbf{k}}{}^{j}$.

If $n \geq 1$ and $\epsilon < \frac{1}{2}$, then $x \in k_x + \frac{1}{2^n} \pm \frac{1}{2}$, which means $x < k_x + 1$. Therefore the guards of the edge leading to $\mathsf{Aut}_{\mathbf{k}}{}^{j}$ is never enabled. Controller can go to B, where it delays $1 + 2\mathsf{frac}(\nu(x))$. Due to perturbations, it will delay $1 + 2\mathsf{frac}(\nu(x)) \pm \delta$ at C. Thus, upon arrival to location D, the valuation satisfies $\mathsf{frac}(z) \in \frac{1}{2^{n-1}} \pm 2\epsilon \pm \delta$. By Lemma 9.5.5, the cost increases by at most $5 + 2\delta$ from here if the play goes into the test module. If $n = 1$, then Controller can go through C' and end in D with $\mathsf{frac}(z) = 2$. Here, the the test module can increase the cost by at most $5 + \epsilon + 2\delta$. □

Complete reduction. To construct the complete reduction, we define for each instruction I_j of the Minsky machine, a module $\mathsf{Instr}_{\mathbf{k}}{}^{j}$ as one of the incrementation or decrementation modules according to the type of I_i. We mark the first location of $\mathsf{Instr}_{\mathbf{k}}{}^{1}$ as the initial location. The halting state $\mathsf{Instr}_{\mathbf{k}}{}^{n}$ of the machine is an accepting location of the timed automaton. For machine M, let $\mathcal{A}_M$ denote the weighted timed automaton constructed in this manner, and let ℓ denote the target location obtained by merging all target locations presented in the above construction.

Theorem 9.3.3 follows from the following proposition.

Proposition 9.5.8. *Minsky machine M halts if, and only if there is a strategy $\sigma \in S_C(\mathcal{G}^{\mathsf{exs}}(\mathcal{A}_M))$ such that $\mathsf{lim\text{-}cost}^{exs}_{\sigma}(\mathcal{A}_M, \ell) \leq 5$.*

Proof. Assume that M halts, say after N steps. Since we are interested in the limit, fix any $\epsilon > 0$ and assume that $0 < 3^N\delta < \epsilon$. We let Controller play respecting the encoding: In modules $\mathsf{Aut}_{\mathbf{k}}{}^{c_i,+}$, it applies the strategy of Lemma 9.5.6 to increment the counter c_i. In modules $\mathsf{Aut}_{\mathbf{k}}{}^{c_i,-}$, it applies the strategy of Lemma 9.5.7 to decrement the counter c_i. Note that this strategy does not depend on δ. At most N modules $\mathsf{Aut}_{\mathbf{k}}{}^{c_i,\cdot}$ are visited since the target location is reached afterwards. Thus, by Lemma 9.5.9, along this play the configuration always belongs to $\mathsf{code}^{\epsilon'}_{\mathbf{k}}(n,m)$ for some $\mathbf{k}$ with $\epsilon' \leq \epsilon$. By these lemmas, either the play simulates the whole path of M and ends in a target location, or it reaches a target sink location (with $\dot{\mathsf{c}} = -1$) earlier. This can be due either to a perturbation by Perturbator along unperturbed edges, or to Perturbator's choice to go to a test module. In both cases the cost is at most $5 + \epsilon + 5\delta \leq 5 + 2\epsilon$, hence the limit is 5.

Assume now that M does not halt. Fix any strategy σ of Controller. If σ respects the encoding, *i.e.* always chooses z as $\frac{\mathsf{frac}(x)}{2}$ or $2\mathsf{frac}(x)$ in $\mathsf{Aut}_{\mathbf{k}c_i,+}$ or $\mathsf{Aut}_{\mathbf{k}c_i,-}$ respectively, then the cost is ∞ if Perturbator never perturbs. Suppose now that σ "cheats" during the simulation at least once. We describe a strategy for Perturbator that leads to a limit cost greater than 5. Perturbator does not perturb the run, and waits until the first instant where Controller does not respect the encoding. This happens inside $\mathsf{Aut}_{\mathbf{k}}{}^{c_i,+}$ or $\mathsf{Aut}_{\mathbf{k}}{}^{c_i,-}$ along a perturbed edge since all other edges are deterministic. Assume for instance the run under strategy σ arrives to location D of $\mathsf{Aut}_{\mathbf{k}c_i,+}$ with $z = 1 + \frac{\mathsf{frac}(x)}{2} + \eta$ for some η depending on σ. Then, Perturbator chooses to go to the test module, which accepts with cost at least $5 + \eta$, by Lemma 9.5.5. Here, η only depends on the strategy σ, so the limit $\mathsf{lim\text{-}cost}^{\mathrm{exs}}_{\sigma}(\mathcal{A}_M)$ is at least $5 + \eta > 5$. □

The following is used in the above proof, to bound the accumulation of the error in the encoding.

Lemma 9.5.9. *Consider the two functions $f : x \mapsto 2x + 1$ and $g : x \mapsto x/2 + 1$. For any $n \geq 1$, $x > 0$, and any $f_1, \ldots, f_n \in \{f, g\}$, $f_1 \circ f_2 \circ \ldots \circ f_n(x) \leq 3^n x$.*

9.6 Lim-OptReach Under Conservative Perturbation

One can argue that the undecidability in the excess perturbation game semantics is due to the ability of Controller to test clock values with precision using equality constraints, and in particular in detecting perturbations. This allows for instance letting Perturbator make a discrete choice, as in the above reduction. Hence, this ability and the possibility of disallowing perturbations on some edges make the semantics of weighted timed automata somehow close to that of two-player weighted timed games in the exact semantics for which the optimal-cost reachability is undecidable.

The conservative perturbation game semantics disallows both abilities. In fact, in this semantics, Controller is required to suggest delays whose perturbations satisfy the guard of the chosen edge. This automatically excludes the use of equality constraints in the guards. Therefore, one cannot encode unperturbed edges as we did with the excess perturbation game semantics. Moreover, the ask-Perturbator module cannot be defined, at least not as previously.

Our result confirms these intuitions: we prove that limit-optimal reachability is PSPACE-complete for weighted timed automata. The algorithm is based on an adaptation of the *corner-point abstraction* restricted to punctual regions. On the other hand, we show that on weighted timed *games*, Lim-OptReach remains undecidable under the conservative perturbation game semantics. More precisely, we show the undecidability of the Lim-OptReach value strict upper bound problem. Our reduction uses the same encoding and similar constructions, but the players will switch roles in ensuring and checking the encoding.

We first present the algorithm for weighted timed automata, and then the undecidability result on weighted timed games.

9.6.1 Algorithm for Weighted Timed Automata

In this section, following [BBBR07], we assume that the clock valuations never go above a given constant, so that all regions are bounded. This is not a loss of generality since we are interested in reachability in weighted timed automata; see [BBBR07].

We recall the corner-point abstraction from [BBBR07], a variant of the region automaton, on which optimal reachability problems can be solved on weighted timed automata.

Definition 9.6.1. *A* finite weighted automaton *is a tuple* (S, s_0, Σ, E, W)*, where* (S, s_0, Σ, E) *is a transition system, with* S*,* Σ *and* E *finite sets, and* $W : E \to \mathbb{Z}$ *is the* weight function.

We will call *paths* the runs of finite weighted automata, to distinguish these from timed automata runs. A path of a finite weighted automaton is defined as a run of the underlying transition system. Intuitively, a finite weighted automaton associates to any path the sum of the weights of the edges it visits. Given any path $\pi = q_1 e_1 q_2 \dots q_n$, the *weight of* π is defined as $W(\pi) = \sum_{1 \le i < n} W(e_i)$.

Let us consider a weighted timed automaton $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E_1, \emptyset, \mathcal{S})$. Notice that we use the notation for weighted timed games; the empty set here is the set of edges for Player 2. The *corner-point abstraction* of $\mathcal{A}$ is a finite weighted automaton, denoted $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$. The states of $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$ are triples (ℓ, r, v), where ℓ is a location, r a region, and $v \in \mathcal{V}(r)$ a vertex of r. Edges are defined as follows: we have $(\ell, r, v) \xrightarrow{\mathsf{delay}} (\ell, r', v')$ if $(\ell, r) \xrightarrow{\mathsf{delay}} (\ell, r')$ in the region automaton, and $v' = v + k$ for some natural number k. In other terms, v' is a time-successor of v, and is a vertex of region r'. The weight associated to this transition is $k \times \mathcal{S}(\ell)$. Further, we have an edge $(\ell, r, v) \xrightarrow{\sigma} (\ell', r', v')$ if there is an edge $(\ell, g, \sigma, R, \ell')$ in $\mathcal{A}$ such that $r \models g$, and $v' = v[R \leftarrow 0]$, $r' = r[R \leftarrow 0]$. Such an edge has weight 0.

Let the *non-punctual corner-point abstraction*, denoted $\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A})$, be the finite weighted automaton obtained from the corner-point abstraction by removing any transition of the form $(\ell, r, v) \xrightarrow{\mathsf{delay}} (\ell, r', v')$, where r' is punctual. Thus, any path in the non-punctual corner-point abstraction corresponds to a non-punctual path in the region automaton.

We define the value of a finite weighted automaton as the cost of the shortest path to a given state. Formally, for any finite weighted automaton $\mathcal{F} = (S, s_0, \Sigma, E, W)$, and state s, let us write

$$\mathsf{value}(\mathcal{F}, s) = \inf_{\substack{\pi \in \mathsf{Runs}(\mathcal{F}) \\ \mathsf{last}(\pi) = s}} W(\pi).$$

For corner-point abstractions, we extend this notation to locations:

$$\mathsf{value}(\mathcal{R}_{\mathsf{cp}}(\mathcal{A}), \ell) = \inf_{\substack{\pi \in \mathsf{Runs}(\mathcal{R}_{\mathsf{cp}}(\mathcal{A})) \\ \mathsf{loc}(\mathsf{last}(\pi)) = \ell}} W(\pi).$$

For any path π of the region automaton $\mathcal{R}(\mathcal{A})$ of $\mathcal{A}$, we denote by $\mathsf{Runs}(\pi)$, the set of runs of $[\![\mathcal{A}]\!]$ that follow π. If π is a path of the corner-point abstraction $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$, then we say that a run follows π if it follows the path projected to $\mathcal{R}(\mathcal{A})$ (that is, obtained by removing vertices in each state). We extend the notation $\mathsf{Runs}(\pi)$ to paths π of the corner-point abstraction.

For any path π of $\mathcal{R}(\mathcal{A})$ or $\mathcal{R}_{\mathsf{cp}}(\mathcal{A})$, let us define $\bar{\pi}$ obtained by replacing all regions by their topological closures. We say that a run follows $\bar{\pi}$ if all its states belong to the closure of the corresponding region in π, and the transitions match those of π.

We recall the following result on optimal cost reachability in the exact semantics.

Lemma 9.6.2 ([BBBR07])**.** *Let* $\mathcal{A}$ *be a weighted timed automaton, and* π *a path in* $\mathcal{R}(\mathcal{A})$*. There exists a run* ρ_0 *that follows* $\bar{\pi}$*, such that for all* $1 \le i \le |\rho|$*,* $\mathsf{state}_i(\rho) \in \mathcal{V}(\mathsf{state}_i(\pi))$*, and*

$$\mathsf{last}(\rho_0)|_c = \inf_{\rho \in \mathsf{Runs}(\pi)} (\mathsf{last}(\rho)|_c).$$

The previous lemma states that the infimum of the cost of the runs that follow a path of the region automaton is reached, in the limit, by a run that only visits the vertices of the regions. Such

a run corresponds to a path of the corner-point abstraction. In fact, the infimum of the cost of the runs is the cost of the shortest path of the corner-point abstraction, as stated in the following lemma.

Lemma 9.6.3 ([BBBR07]). *For any weighted timed automaton $\mathcal{A}$ and target location ℓ,*

$$\mathsf{value}(\mathcal{A}, \ell) = \mathsf{value}(\mathcal{R}_{cp}(\mathcal{A}), \ell).$$

In the perturbed case, we prove that the same algorithm can be applied, once we discard punctual paths.

Lemma 9.6.4. *Given any weighted timed automaton $\mathcal{A}$ and target location ℓ,*

$$\mathsf{lim\text{-}value}^{cons}(\mathcal{A}, \ell) = \mathsf{value}(\mathcal{R}^{np}_{cp}(\mathcal{A}), \ell).$$

Hence to compute the optimal cost on $\mathcal{A}$, it suffices to consider the finite weighted automaton $\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A})$, and find the shortest path to location ℓ. To decide whether the limit value is less than some given constant λ, one can guess a path in $\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A})$ in polynomial-space, and check whether its weight is less or equal to than λ. Note that the problem is PSPACE-hard since it is already in the unweighted case. Theorem 9.3.4 follows.

The following lemma says that paths of the corner-point abstraction can be approximated by valid runs; this is used in the proof of Lemma 9.6.4,

Lemma 9.6.5 ([BBBR07]). *Let π be a path of $\mathcal{R}_{cp}(\mathcal{A})$. For any $\epsilon > 0$, there exists a run ρ_ϵ that follows π such that $\mathsf{last}(\rho_\epsilon)|_c \leq W(\pi) + \epsilon$.*

The following simple lemma states that an infinitesimal perturbation suffices to go from a punctual region to a non-punctual one.

Lemma 9.6.6. *Given any $\delta > 0$, and $\nu \in r$ in a punctual region r, there exists $\epsilon \in [-\delta, \delta]$ such that $\mathsf{reg}(\nu + \epsilon)$ is non-punctual.*

We now prove Lemma 9.6.4.

Proof of Lemma 9.6.4. We will prove the two inequalities. First, it is easy to show that $\mathsf{lim\text{-}value}^{\mathsf{cons}}(\mathcal{A}, \ell) \geq \mathsf{value}(\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A}), \ell)$. In fact, let $\sigma \in S_C(\mathcal{G}^{\mathsf{cons}}_\delta(\mathcal{A}))$ be any strategy for Controller. We consider some strategy σ' for Perturbator, that perturbs so as to render the run non-punctual. More precisely, if the delay suggested by Controller is inside a non-punctual region, Perturbator plays 0, and otherwise it plays some small non-zero amount, by Lemma 9.6.6 below. Then the projection of the outcome of (σ, σ') to regions is a non-punctual initialized path π of $\mathcal{R}(\mathcal{A})$. But by Lemma 9.6.2, the cost of this run cannot be smaller than some run along $\bar{\pi}$ visiting vertices, and for any such run, there is a path in $\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A})$ with the same cost.

To prove the other direction, for any $\epsilon > 0$, we will construct a strategy for Controller in $\mathcal{A}$ that achieves cost $\mathsf{value}(\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A}), \ell) + O(\epsilon + \delta)$, so this will be equal to $\mathsf{value}(\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A}), \ell) + O(\epsilon)$ in the limit, hence the infimum is $\mathsf{value}(\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A}), \ell)$ as required. Assume that $\mathsf{value}(\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A}), \ell)$ is finite. Let π denote a non-punctual path of $\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A})$ with $W(\pi) = \mathsf{value}(\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A}), \ell)$. By Lemma 9.6.5, there exists a run ρ_ϵ in $\mathcal{A}$ that follows π, with $\mathsf{last}(\rho_\epsilon)|_{\mathsf{c}} \leq W(\pi) + \epsilon$. If all delays in ρ_ϵ are positive, then let $\epsilon_0 > 0$ smaller than any delay and such that all timed traces $\mathsf{Ball}_d(\mathsf{ttrace}(\rho_\epsilon), \epsilon_0)$ induce valid runs that follow π. Here, such an $\epsilon_0 > 0$ exists since $\mathsf{ttrace}(\pi)$ is an open set (Proposition 8.3.4). Let $\delta \leq \epsilon_0$. Then, in $\mathcal{G}^{\mathsf{cons}}_\delta(\mathcal{A})$, if Controller suggests the delays and actions of $\mathsf{ttrace}(\pi)$, any

outcome ρ is a valid run that follows π. Moreover, since each delay can be perturbed at most by δ, we have $\mathsf{last}(\rho)|_{\mathsf{c}} \leq \mathsf{last}(\rho_\epsilon)|_{\mathsf{c}} + \delta|\pi|S \leq W(\pi) + \epsilon + \delta|\pi|S$, where S is the maximal slope of the cost at locations visited by π. The lemma follows.

If not all delays of ρ_ϵ are positive, then we choose another run ρ'_ϵ with this property, such that $d(\mathsf{ttrace}(\rho_\epsilon), \mathsf{ttrace}(\rho'_\epsilon)) \leq \epsilon$. This is possible since $\mathsf{ttrace}(\pi)$ is open. In this case, the above proof yields a run ρ with $\mathsf{last}(\rho)|_{\mathsf{c}} \leq W(\pi) + 2\epsilon + \delta|\pi|S$.

If $\mathsf{value}(\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A}), \ell) = \infty$, then ℓ is not reachable by any run, so $\mathsf{lim\text{-}value}^{\mathrm{cons}}(\mathcal{A}, \ell) = \infty$. If $\mathsf{value}(\mathcal{R}^{\mathsf{np}}_{\mathsf{cp}}(\mathcal{A}), \ell) = -\infty$, then one can construct the above strategy for any $M < 0$, considering a path π with $W(\pi) < M$, which yields $\mathsf{lim\text{-}value}^{\mathrm{cons}}(\mathcal{A}, \ell) = -\infty$. □

9.6.2 Undecidability on Weighted Timed Games

We prove Theorems 9.3.5 and 9.3.6 using a reduction similar to the one in Section 9.5. In this section, strategies and outcomes refer to the conservative perturbation game semantics.

Test Module. We define the new module $\mathsf{Add}'^{1+x}_{\mathbf{k}}(x, u, t)$ depicted in Fig. 9.8. We do not have any guards on the edges; Controller checks at the last step whether all delays were chosen as expected. Recall that the dotted edges are uncontrollable edges, so they are taken by Perturbator. Here, there is no upper bound on the delays that can be suggested by Perturbator, so the cost can increase arbitrarily. However, the target sink state has a negative slope on variable c, so the cost is $-\infty$ whenever Perturbator "cheats".

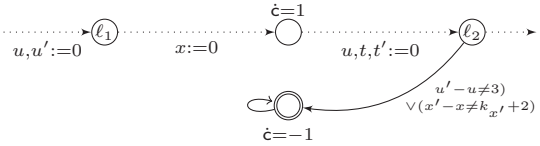


Figure 9.8: Module $\mathsf{Add}'^{1+x}_{\mathbf{k}}(x, u, t)$.

We define similarly modules $\mathsf{Add}'^{2-x}_{\mathbf{k}}(x, u, t)$. As previously, these modules can be combined to define $\mathsf{Add}'^{5+x-2z}_{\mathbf{k}}(x, z, u, t)$. Since it is now Perturbator's duty to respect the encoding, Controller will test the chosen values using the $\mathsf{Test}'^{x=2z}_{\mathbf{k}}(x, z, u, t)$ module, in Fig. 9.9.

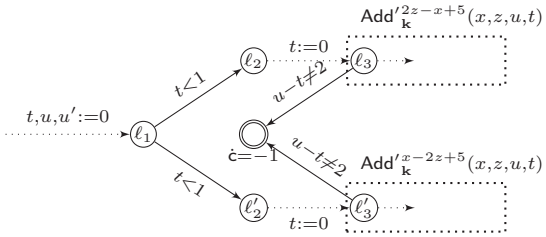


Figure 9.9: Module $\mathsf{Test}'^{x=2z}_{\mathbf{k}}(x, z, u, t)$.

Lemma 9.6.7. *Assume that module* $\mathsf{Test}'^{x=2z}_{\mathbf{k}}(x,z,u,t)$ *is entered at* ℓ_1 *with valuation* $\nu \in \mathsf{code}^{\epsilon}_{\mathbf{k}}$ *for some* $0 < \epsilon < \frac{1}{2}$*. Then, Controller has a strategy to reach a target location with cost at most* $5 - |\mathsf{frac}(\nu(x)) - 2\mathsf{frac}(\nu(z))|$.

Incrementation Module $\mathsf{Aut}'^{c_i,+}_{\mathbf{k}}$. Now, the module $\mathsf{Aut}'^{c_1,+}_{\mathbf{k}}(x,y,z,u,t)$ that increments counter c_1, given in Fig. 9.6, is defined similarly to the excess-perturbation case, by exchanging the roles of the players. Along this module, if Perturbator does not get the right value for z (which is $z = 1 + \mathsf{frac}(x)/2$), then Controller can go to the test module and reach the target location with cost $5 - |2\mathsf{frac}(z) - \mathsf{frac}(x)|$; it can also continue to $\mathsf{Instr}'^{j}_{\mathbf{k}}$. We omit the definition of $\mathsf{Reduce}_{\mathbf{k}'}$ but it can be adapted from $\mathsf{Reduce}_{\mathbf{k}}$ without difficulty, similarly to $\mathsf{Add}'_{\mathbf{k}}$ modules: Perturbator determines the run, and Controller checks it afterwards. The module $\mathsf{Aut}'^{c_2,+}_{\mathbf{k}}(x,y,z,u,t)$ is defined similarly by exchanging the roles of x and y.

We state the properties of module $\mathsf{Aut}'^{c_1,+}_{\mathbf{k}}(x,y,z,u,t)$ more generally since we would like Controller to tolerate a bounded error in Perturbator's move:

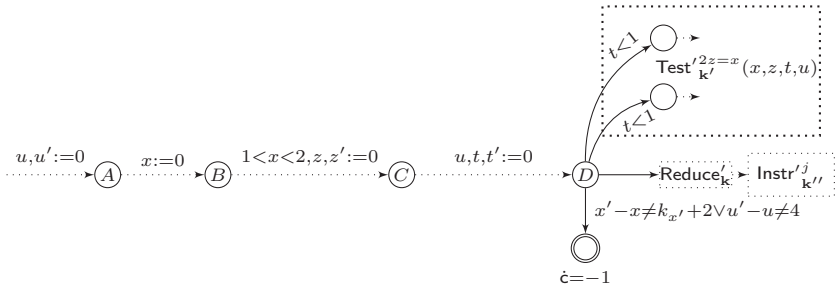


Figure 9.10: Module $\mathsf{Aut}'^{c_1,+}_{\mathbf{k}}(x,y,z,u,t)$. Here $\mathsf{Reduce}_{\mathbf{k}'}$ refers to $\mathsf{Reduce}_{\mathbf{k}'}(z,y,u,t)$ and $\mathsf{Instr}'^{j}_{\mathbf{k}''}$ to $\mathsf{Instr}'^{j}_{\mathbf{k}''}(z,y,x,u,t)$.

Lemma 9.6.8. *Assume that module* $\mathsf{Aut}'^{c_1,+}_{\mathbf{k}}(x,y,z,u,t)$ *is entered at* A *with valuation* $v \in \mathsf{code}^{\epsilon}_{\mathbf{k}}(n,m)$ *with* $\epsilon < \frac{1}{4}$*. For any* $\eta > 0$*, Controller has a strategy to either reach a target location with cost at most* $5 - \eta$*, or reach* $\mathsf{Instr}'^{j}_{\mathbf{k}}$ *with a valuation* $\nu' \in \mathsf{code}^{\epsilon/2+\eta}_{\mathbf{k}'}(n+1,m)$.

Proof. Fix $\eta > 0$. The strategy is the following. If upon arrival to D, the valuation ν' satisfies $|2\mathsf{frac}(\nu'(z)) - \mathsf{frac}(\nu(x))| > \eta$, then Controller goes to the test module. Otherwise, it proceeds to $\mathsf{Instr}'^{j}_{\mathbf{k}}$ after going through the $\mathsf{Reduce}'_{\mathbf{k}}$. □

Decrementation Module $\mathsf{Aut}'^{c_1,-}_{\mathbf{k}}$. The module $\mathsf{Aut}'^{c_1,-}_{\mathbf{k}}(x,y,z,u,t)$, given in Fig. 9.11, is also an adaptation of $\mathsf{Aut}'^{c_1,-}_{\mathbf{k}}(x,y,z,u,t)$ of the excess-perturbation case.

Lemma 9.6.9. *Assume module* $\mathsf{Aut}'^{c_1,-}_{\mathbf{k}}$ *is entered with valuation* $\nu \in \mathsf{code}^{\epsilon}_{\mathbf{k}}(n,m)$ *and cost* 0*, with* $\epsilon < \frac{1}{4}$*. Then,*

- *If* $n = 0$*, then Controller has a strategy to ensure reaching either* $\mathsf{Instr}'^{j}_{\mathbf{k}}$ *with the same configuration and cost, or an accepting location with cost* $-\infty$*. In this case, Controller cannot reach* $\mathsf{Instr}'^{j'}_{\mathbf{k}''}$.

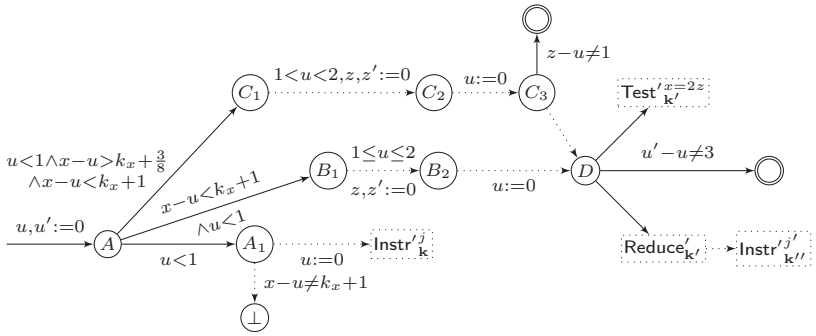


Figure 9.11: Module $\mathsf{Aut}'^{\,c_1,-}_{\mathbf{k}}(x, y, z, u, t)$: decrementation with zero test.

- *If $n \geq 1$, then Controller cannot reach $\mathsf{Instr}'^{j}_{\mathbf{k}}$.*
- *If $n \geq 1$, for any $\eta > 0$, Controller has a strategy that ensures that either $\mathsf{Instr}'^{j'}_{\mathbf{k}''}$ is reached with valuation $\nu' \in \mathsf{code}^{2\epsilon+\eta}_{\mathbf{k}'}(n-1, m)$ for some $\mathbf{k}'$, or the target location is reached with cost at most $5 - \eta$.*

Proof. Recall that, by definition, for any $\nu \in \mathsf{code}^{\epsilon}_{\mathbf{k}}(0, m)$, $\nu(x) = k_x$. The module $\mathsf{Instr}'^{j}_{\mathbf{k}}$ is reachable if, and only if $n = 0$ since Perturbator can lead to the non-accepting sink state ($\bot$) whenever Controller moves to A_1 while $\nu(x) \neq k_x$; otherwise $\mathsf{Instr}'^{j}_{\mathbf{k}}$ is reached. The guard of the other edges from A are not satisfied since $x - u = k_x + 1$.

If $n = 1$, Controller can go to C_1. In fact, since $\epsilon < \frac{1}{4}$, the guard $x - u > k_x + \frac{3}{8}$ is satisfied. Then, the path from C_1 to D forces Perturbator to ensure $z - u = 1$. If Controller goes to $\mathsf{Instr}'^{j'}_{\mathbf{k}''}$, this condition yields a valuation in $\mathsf{code}^{\epsilon}_{\mathbf{k}''}(0, m)$. If $n \neq 1$, then the guard of the edge from A to C_1 is not satisfied by the assumption on ϵ.

When $n \geq 1$, Controller goes to B_1, from where Perturbator is expected to lead to D with some valuation ν' satisfying $\nu'(z) = 2\mathsf{frac}(\nu(x)) \pm \alpha$, for some α. Now, if $|\alpha| \geq \eta$, we let Controller go to the test module, therefore, end in an accepting location with cost at most $5 - \eta$ (by Lemma 9.6.7). Otherwise, it proceeds to $\mathsf{Instr}'^{j'}_{\mathbf{k}''}$. □

Complete Reduction. Let $\mathcal{A}'_M$ denote the weighted timed game constructed in this manner for a machine M. Let ℓ denote the target location obtained by merging all target location of the above modules. Theorems 9.3.5 and 9.3.6 follow from the following proposition.

Proposition 9.6.10. *Minsky machine M halts if, and only if*

$$\inf_{\sigma \in S_C(\mathcal{G}^{\mathsf{cons}}_{\delta}(\mathcal{A}))} \mathit{lim\text{-}cost}^{cons}_{\sigma}(\mathcal{A}'_M, \ell) < 5,$$

if, and only if for all $\delta \in [0, \frac{1}{3}]$,

$$\inf_{\sigma \in S_C(\mathcal{G}^{\mathsf{cons}}_{\delta}(\mathcal{A}))} \sup_{\sigma' \in S_P(\mathcal{G}^{\mathsf{cons}}_{\delta}(\mathcal{A}))} \mathit{cost}^{\ell}_{\sigma,\sigma'}(\mathcal{A}'_M) < 5,$$

if, and only if for all $\delta \in [0, \frac{1}{3}]$,

$$\inf_{\sigma \in S_C(\mathcal{G}_\delta^{\mathbf{exs}}(\mathcal{A}))} \sup_{\sigma' \in S_P(\mathcal{G}_\delta^{\mathbf{exs}}(\mathcal{A}))} \mathsf{cost}^{\ell}_{\sigma,\sigma'}(\mathcal{A}'_M) < 5,$$

Proof. Assume that the machine halts, say in N steps. For any $0 < \epsilon < \frac{1}{4}$, we fix $0 < \eta < \epsilon \times 3^{-N}$, and assume that $0 \leq \delta \leq \eta$. Controller follows the strategy given by Lemmas 9.6.8 and 9.6.9, respectively, in modules $\mathsf{Aut}'^{\,c_i,+}_{\mathbf{k}}$ and $\mathsf{Aut}'^{\,c_i,-}_{\mathbf{k}}$. Thus, Controller goes to the test module if and only if the error in the incrementation or decrementation is more than η, otherwise it continues the simulation. Then by Lemma 9.5.9, along any play, upon arrival to these modules the valuation belongs to $\mathsf{code}^{\epsilon}_{\mathbf{k}}(n, m)$ for some n, m. Thus, Controller reaches the target location with cost at most $5 - \eta$. Note that δ has no effect here as long as it is less than $\frac{1}{2}$. In fact, the only edges that belong to Controller either yield to target locations, or the amounts of delays are irrelevant since exact timing is ensured by Perturbator in subsequent transitions.

Assume now that the machine does not halt. In this case, for any strategy of Controller, we let Perturbator respect the encoding exactly. The play is then either never reaches a target location, or it goes inside a test module and ends in a target location with cost 5. □

9.7 Conclusion

In this chapter, we considered perturbation game semantics in weighted timed automata and games. The only decidable variant of the problem is the cost optimal reachability in weighted timed automata under the conservative perturbation game semantics, where PSPACE-completeness is established by adapting the corner-point abstraction. We also showed "robust undecidability" results for weighted timed games: optimal reachability problems remain undecidable under perturbation game semantics. Moreover, the problem even becomes undecidable for weighted timed automata in the excess perturbation game semantics.

The latter result is particularly surprising. In fact, in weighted timed automata, the decidability of the cost-optimal reachability without perturbations can be proved using the corner-point abstraction. Lemma 9.6.5 shows that runs with (approximately) optimal costs follow the paths in this construction by staying close, by some ϵ, to the vertices of a given path. In Chapter 7, we used similar ideas to obtain the decidability of the parameterized robust reachability in the excess perturbation game semantics. Our construction extends region automata by shrinking constraints, which mark subsets of facets of each region (rather than marking vertices, as in the corner-point abstraction). Our proof shows that Controller cannot avoid being close to these marked facets. Thus, although both techniques are similar, our undecidability result shows, perhaps counterintuitively, that the two techniques cannot be combined to construct an algorithm.

The encodings yielding undecidability are due, in both cases, to the ability of either of the players to play precisely, and the other one to check previous delays with precision. We conclude that game semantics do not introduce enough "fuzziness" in the semantics to avoid encoding undecidable languages.

The problems we have studied have several variants, and we did not study all here: the strategy upper bound problem for the conservative perturbation game semantics, and the strategy strict upper bound for the excess perturbation game semantics remain open. Furthermore, the decidability of the upper bounds on the *value* is also open for weighted timed games. We conjecture that these problems should be undecidable on weighted timed games. On the other hand, restricting to closed

guards might yield to decidability, since then players would not be able to check the nonequality of the clock values.

Part IV

Robust Implementation

In this part, we present results on the robust implementation problem. Here, the goal is to transform a given timed automaton into one that is equivalent to the original one, in a precise sense, but also robust. Hence, in the robust implementation approach, a timed automaton model can be designed and checked using the exact semantics, and then (automatically) robustly implemented. We present two kinds of results. In Chapter 10, we give procedures to transform any timed automaton into one whose enlargement is approximately bisimilar to the original automaton. The same result holds for the sampling semantics. In Chapter 11, we study a concrete semantics for timed automata with a discrete clock, communication and synchronization delays. We prove that shrinkable timed automata preserve their behaviors, in the sense of time-abstract simulation equivalence and non-blockingness, under this semantics.

Chapter 10

Approximate Implementation

10.1 Introduction

We saw in Chapter 2 how new behaviours appear in timed automata under enlargement, while behaviours disappear under sampling. In this chapter, we investigate the possibility of modifying a given timed automaton so as to avoid robustness problems; we would like their enlarged and sampled semantics to preserve their exact semantics. Given *any* timed automaton $\mathcal{A}$, we build another timed automaton $\mathcal{B}$ whose semantics under enlargement and under sampling is *approximately bisimilar* to $\mathcal{A}$. Approximate bisimilarity is formalized by a quantitative variant of bisimulation from [FLT10] where the differences between the timings in two systems are bounded above by a parameter ϵ. Our construction is parameterized and provides a bisimilar implementation for any desired precision $\epsilon > 0$.

Our results mean that all timed automata can be approximately implemented under guard enlargement or sampling. This provides a robust refinement / implementation procedure: one can design timed automata under the exact semantics, and then automatically obtain a correct-by-construction implementation. Our results have also a theoretical implication: In Section 10.5, we show that several undecidable problems on timed automata remain undecidable for the class of robust timed automata.

The results of this chapter were published in [BLM+11].

10.2 Preliminaries

10.2.1 Behaviourial Relations

We consider a quantitative extension of timed bisimilarity considered in [TFL10]. This spans the gap between timed and time-abstract bisimulations: while the former requires time delays to be matched exactly, the latter ignores timing information altogether. Intuitively, we define two states to be ϵ-bisimilar, for a given parameter $\epsilon \geq 0$, if there is a (time-abstract) bisimulation which relates these states in such a way that, at each step, the difference between the time delays of corresponding delay transitions is at most ϵ. Thus, this parameter allows one to quantify the "timing error" made during the bisimulation. A strong and a weak variant of this notion is given in the following definition.

Definition 10.2.1. *Given a TTS $(S, s_0, \Sigma, \mathbb{K}, \to)$, and $\epsilon \geq 0$, a symmetric relation $R_\epsilon \subseteq S \times S$ is a*

- strong timed ϵ-bisimulation*, if for any $(s,t) \in R_\epsilon$ and $\sigma \in \Sigma, d \in \mathbb{K}$,*
  - *$s \xrightarrow{\sigma} s'$ implies $t \xrightarrow{\sigma} t'$ for some $t' \in S$ with $(s',t') \in R_\epsilon$,*
  - *$s \xrightarrow{d} s'$ implies $t \xrightarrow{d'} t'$ for some $t' \in S$ and $d' \in \mathbb{K}$ with $|d - d'| \leq \epsilon$ and $(s',t') \in R_\epsilon$.*

- timed-action ϵ-bisimulation, *if for any* $(s,t) \in R_\epsilon$, *and* $\sigma \in \Sigma$, $d \in \mathbb{K}$,
  - $s \xrightarrow{d,\sigma} s'$ *implies* $t \xrightarrow{d',\sigma} t'$ *for some* $t' \in S$ *and* $d' \in \mathbb{K}$ *with* $|d - d'| \leq \epsilon$ *and* $(s',t') \in R_\epsilon$.

If there exists a strong timed ϵ-bisimulation (resp. timed-action ϵ-bisimulation) R_ϵ such that $(s,t) \in R_\epsilon$, then we write $s \sim_\epsilon t$ (resp. $s \approx_\epsilon t$). Furthermore we write $s \sim_{\epsilon^+} t$ (resp. $s \approx_{\epsilon^+} t$) whenever for every $\epsilon' > \epsilon$, $s \sim_{\epsilon'} t$ (resp. $s \approx_{\epsilon'} t$).

Observe that $s \sim_\epsilon t$ implies $s \sim_{\epsilon'} t$ for every $\epsilon' > \epsilon$. Also, $s \sim_{\epsilon^+} t$ does not imply $s \sim_\epsilon t$ in general (see Fig. 10.1), and if $s \sim_{\epsilon^+} t$ but $s \not\sim_\epsilon t$, then $\epsilon = \inf\{\epsilon' > 0 \mid s \sim_{\epsilon'} t\}$. These observations hold true in the timed-action bisimulation setting as well. Note also that $s \sim_\epsilon t$ implies $s \approx_\epsilon t$. Finally, for $\epsilon > 0$, strong timed or timed-action ϵ-bisimilarity relations are not equivalence relations in general, but they are when $\epsilon = 0$.

Figure 10.1: An automaton in which $(s,0) \sim_{0^+} (t,0)$ but $(s,0) \not\sim_0 (t,0)$.

Last, we define a variant of *ready-simulation* [LS89] for timed transition systems. For $\mathsf{Bad} \subseteq \Sigma$, we will write $I \sqsubseteq^{\mathsf{Bad}} S$ when I is simulated by S (and time delays are matched exactly) in such a way that at any time during the simulation, any failure (*i.e.*, any action in Bad) enabled in S is also enabled in I. So, if $I \sqsubseteq^{\mathsf{Bad}} S$ and S is safe w.r.t. Bad (*i.e.*, Bad actions are never enabled), then any run of I can be executed in S (with exact timings) without enabling any of the Bad-actions. Fig. 10.2 shows an automaton illustrating the importance of this notion. More formally:

Definition 10.2.2. *Given a TTS $(S, s_0, \Sigma, \mathbb{K}, \to)$, and a set $\mathsf{Bad} \subseteq \Sigma$, a relation $R \subseteq S \times S$ is a* ready-simulation w.r.t. Bad *if, whenever $(s,t) \in R$:*

- *for all $\sigma \in \Sigma$ and $d \in \mathbb{K}$, $s \xrightarrow{d,\sigma} s'$ implies $t \xrightarrow{d,\sigma} t'$ for some $t' \in S$ with $(s',t') \in R$,*
- *for all $\sigma \in \mathsf{Bad}$, $t \stackrel{\sigma}{\Rightarrow} t'$ implies $s \stackrel{\sigma}{\Rightarrow} s'$ for some $s' \in S$.*

We write $s \sqsubseteq^{\mathsf{Bad}} t$ if $(s,t) \in R$ for some ready-simulation R w.r.t. Bad.

We write $[\![\mathcal{A}]\!] \sim_\epsilon [\![\mathcal{B}]\!]$, $[\![\mathcal{A}]\!] \approx_\epsilon [\![\mathcal{B}]\!]$ and $[\![\mathcal{A}]\!] \sqsubseteq^{\mathsf{Bad}} [\![\mathcal{B}]\!]$ when the initial states of timed automata $\mathcal{A}$ and $\mathcal{B}$ are related accordingly in the disjoint union of the transition systems.

10.2.2 Refined Regions

Let $\mathbb{Q}_\infty = \mathbb{Q} \cup \{-\infty, \infty\}$. For $M, \eta \in \mathbb{Q}_{>0}$ such that $\frac{1}{\eta} \in \mathbb{N}$, we denote by $\Phi_{\mathcal{C}}(\eta, M)$ the set of guards on the clock set $\mathcal{C}$, whose constants are either $\pm\infty$ or less than or equal to M in absolute value and are integer multiples of η. Remember that $\Phi_{\mathcal{C}}$ denotes the set of all guards on clock set $\mathcal{C}$. We call the inverses of positive integers *granularities*. The *granularity* of a timed automaton is the inverse of the least common denominator of the finite constants in its guards.

We will define a a more general version of the regions defined in Chapter 2. Observe that region equivalence could also be defined by relating valuations that cannot be distinguished by (some subset of) guards. In fact, all conditions defining regions, such as fractional ordering, equality of integer

parts can be encoded using guards. Here, we will use this approach rather than explicitly defining the (refined) region equivalence.

Let M be the maximum (rational) constant that appears in the guards of $\mathcal{A}$, let η be the granularity of $\mathcal{A}$. Multiplying any constant in $\mathcal{A}$ by $\frac{1}{\eta}$, we obtain an integral timed automaton. Given valuations $u, v \in \mathbb{R}^{\mathcal{C}}_{\geq 0}$ and rationals M, η, define $v \simeq^M_\eta u$ to hold if, and only if, for all formulas $\phi \in \Phi_{\mathcal{C}}(\eta, M)$, $u \models \phi$ if and only if $v \models \phi$. The equivalence class of a valuation u for the relation $\simeq^M_\eta$ is denoted by $\mathsf{reg}(u)^M_\eta = \{v \mid u \simeq^M_\eta v\}$. Each such class is called an (η, M)-region. In the rest, when constant M is (resp. M and η are) clear from context, we simply write $\mathsf{reg}(u)_\eta$ (resp. $\mathsf{reg}(u)$) and call these η-regions (resp. regions). We denote by $\overline{\mathsf{reg}(u)^M_\eta}$ the topological closure of $\mathsf{reg}(u)^M_\eta$, for the usual topology on $\mathbb{R}^{\mathcal{C}}$. The number of (η, M)-regions is bounded by $O(2^{|\mathcal{C}|}|\mathcal{C}|!(M/\eta)^{|\mathcal{C}|})$ [AD94].

In our definition of regions, we thus treat $\frac{1}{M}$ as the unit value. Notice that if all constants are integers, then we have $M = 1$ and our definition gives the usual notion of region equivalence. To understand the effect of M, consider valuations $u = (0.5, 0.5)$ and $v = (0.4, 0.4)$. Let $K \geq 1$. We have $\mathsf{reg}(u)_1 = \mathsf{reg}(v)_1$, since this is the usual notion of region equivalence. However, $\mathsf{reg}(u)_2 \neq \mathsf{reg}(v)_2$ since the formula $x < 0.5$ distinguishes these valuations. It is clear that 1-regions are unions of M-regions for any $M \geq 1$.

We define $\mathsf{tsucc}^*(r)$ as the set of *time-successor regions of* r, that is, the set of η-regions r' such that $u + d \in r'$ for some $u \in r$ and $d \in \mathbb{R}_{\geq 0}$.

We now associate with each (η, M)-region r a guard ϕ_r in $\Phi(M, \eta)$ that defines r. We assume ϕ_r to be normalized, that is, the DBM corresponding to ϕ_r is normalized (see Section 3.3).

10.3 Implementability

In this chapter, we use the term *robustness* to refer to behaviour preservation under guard enlargement, while *samplability* refers to behaviour preservation under sampling of time. We give several definitions which distinguish timed automata whose enlargement (resp. whose sampled semantics) is ϵ-bisimilar to the original automaton, for some ϵ.

10.3.1 Robustness

The approach taken in Chapter 4 and earlier work [Pur00, DDMR08] concentrated on comparing timed automata $\mathcal{A}$ and some enlargement $\mathcal{A}_\Delta$ with respect to untimed language equivalence, or the satisfaction of a given logic formula. Here, we consider a stronger notion of robustness which requires two systems to be ϵ-bisimilar for some ϵ.

Definition 10.3.1. *A timed automaton $\mathcal{A}$ is* ϵ-bisimulation-robust *(or simply ϵ-robust), where $\epsilon \geq 0$, if there exists $\Delta > 0$ such that $[\![\mathcal{A}]\!] \approx_\epsilon [\![\mathcal{A}_\Delta]\!]$.*

Clearly, not all timed automata are robust in this sense, since the timed automaton $\mathcal{A}^2$ of Fig. 2.2 is not robust even for location reachability.

We do not know whether a timed automaton that is robust for some Δ is still robust for any $\Delta' < \Delta$, that is, whether $[\![\mathcal{A}]\!] \approx_\epsilon [\![\mathcal{A}_\Delta]\!]$ implies $[\![\mathcal{A}]\!] \approx_\epsilon [\![\mathcal{A}_{\Delta'}]\!]$ for $\Delta' < \Delta$, in general. This is the so-called "faster-is-better" property [AT05, DDR05a], which means that if a property holds in some platform, it also holds in a faster or more precise platform. This is known to be satisfied for simpler notions of robustness mentioned above.

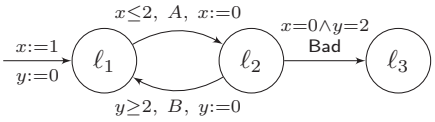


Figure 10.2: A copy of the non-robust timed automaton $\mathcal{A}^2$ seen earlier in Fig. 2.2

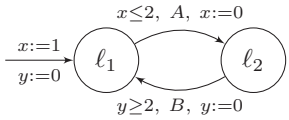


Figure 10.3: A robust but unsafe alternative.

Bisimulation is not always sufficient for refining systems when one wants to preserve state-based safety properties proven for $\mathcal{A}$. For instance, removing edges leading to unsafe states in $\mathcal{A}$ may provide us with a trivially safe automaton under any enlargement. However, edges leading to such states are used to detect failures, so removing these will not necessarily remove the failure itself (since the states that immediately trigger a failure may still be reachable). Fig. 10.3 gives such an "incorrect" construction. To cope with this problem, we rely on ready-simulation and require $\mathcal{A}'$ to satisfy $[\![\mathcal{A}'_\Delta]\!] \sqsubseteq^{\mathsf{Bad}} [\![\mathcal{A}_\Delta]\!]$, where Bad are distinguished actions leading to unsafe states in $\mathcal{A}$. This means that any run of $[\![\mathcal{A}'_\Delta]\!]$ can be realized in $[\![\mathcal{A}_\Delta]\!]$, and that no Bad-action is enabled in that run in the latter (and hence no unsafe state is reached). Thus, intuitively, no state reached in $[\![\mathcal{A}'_\Delta]\!]$ corresponds to an unsafe state in $[\![\mathcal{A}_\Delta]\!]$, and in particular, if $\mathcal{A}_\Delta$ has unsafe runs (leading to unsafe states), then these cannot be realized in $\mathcal{A}'_\Delta$. Clearly, the automaton in Fig. 10.3 does not satisfy this. We formalize this idea here.

Definition 10.3.2. *A timed automaton $\mathcal{A}$ is* safe w.r.t. a set of actions $\mathsf{Bad} \subseteq \Sigma$, *if $d_\infty\big(\mathit{Reach}([\![\mathcal{A}]\!]), \mathit{Pre}(\mathit{Bad})\big) > 0$, where d_∞ is the standard infinity metric, and $\mathit{Pre}(\mathit{Bad}) = \bigcup_{\sigma\in\mathit{Bad},(\ell,g,\sigma,R,\ell')\in E} \{\ell\} \times [\![g]\!]$ is the precondition of Bad-actions*

Here, $\mathit{Pre}(\mathsf{Bad})$ is the set of states from which a Bad action can be done. Note that $d_\infty(\mathsf{Reach}([\![\mathcal{A}]\!]), \mathit{Pre}(\mathsf{Bad})) = 0$ does not imply that a state of $\mathit{Pre}(\mathsf{Bad})$ is reachable in $\mathcal{A}$. But we still consider such an automaton as unsafe, since, intuitively, any enlargement of the guards may lead to a state of $\mathit{Pre}(\mathsf{Bad})$. It can be seen that automaton of Fig. 10.2 is safe w.r.t. action Bad. Note that a closed timed automaton is safe w.r.t. Bad if, and only if Bad is not reachable, since the state space is topologically closed [GHJ97].

We also define a robustness notion requiring only safety properties, as in [Pur00, DDMR08].

Definition 10.3.3. *A timed automaton $\mathcal{A}$ is* safety-robust (w.r.t. Bad) *if there exists $\Delta > 0$ such that $[\![\mathcal{A}_\Delta]\!]$ is safe w.r.t. Bad.*

In the rest, Bad will refer to a set of actions given with the timed automaton we consider. When we say that a timed automaton is safe, or safety-robust, these actions will sometimes be implicit.

We introduce the notion of *safety-robust implementation* (parameterized by a bisimilarity relation $\equiv$, which will range over $\{\sim_0, \sim_{0^+}, \approx_0, \approx_{0^+}\}$), where we only require the alternative automaton to preserve a given safety specification.

Definition 10.3.4 (Safety-Robust Implementation)**.** *Let $\mathcal{A}$ be a timed automaton which is safe w.r.t. actions Bad, and $\equiv$ denote any bisimilarity relation. A* safety-robust implementation of $\mathcal{A}$ w.r.t $\equiv$ *is a timed automaton $\mathcal{A}'$ such that:*

(*i*) *$\mathcal{A}'$ is safety-robust;*

(*ii*) $[\![\mathcal{A}']\!] \equiv [\![\mathcal{A}]\!]$;

(*iii*) *there exists* $\Delta_0 > 0$ *s.t. for all* $0 < \Delta' < \Delta < \Delta_0$, $[\![\mathcal{A}'_{\Delta'}]\!] \sqsubseteq^{\mathsf{Bad}} [\![\mathcal{A}_\Delta]\!]$.

Now we define the notion of *robust implementation*. We require such an implementation to be robust and equivalent to the original automaton, and to preserve safety specifications.

Definition 10.3.5 (Robust Implementation)**.** *Let* $\mathcal{A}$ *be a timed automaton which is safe w.r.t. actions* Bad*, and* $\equiv$ *denote any bisimilarity. An* ϵ-robust implementation of $\mathcal{A}$ w.r.t. $\equiv$ *is a timed automaton* $\mathcal{A}'$ *such that:*

(*i*) $\mathcal{A}'$ *is* ϵ*-robust;*

(*ii*) $[\![\mathcal{A}']\!] \equiv [\![\mathcal{A}]\!]$;

(*iii*) *there exists* $\Delta_0 > 0$ *s.t. for all* $0 < \Delta' < \Delta < \Delta_0$, $[\![\mathcal{A}'_{\Delta'}]\!] \sqsubseteq^{\mathsf{Bad}} [\![\mathcal{A}_\Delta]\!]$.

10.3.2 Samplability

We now consider the sampled semantics, and define *samplability*. Similarly to the robustness definitions, we are interested in approximate bisimulation between the exact semantics and the sampled one.

Definition 10.3.6. *A timed automaton is said to be* ϵ-bisimulation-samplable *(or simply* ϵ-samplable*) if there exists a granularity* η *such that* $[\![\mathcal{A}]\!] \approx_\epsilon [\![\mathcal{A}]\!]^\eta$.

Note that not all timed automata are bisimulation-samplable: [KP05] describes timed automata $\mathcal{A}$ which are not (time-abstract) bisimilar to their sampled semantics for any granularity η. We define a *sampled implementation* as follows.

Definition 10.3.7 (Sampled Implementation)**.** *Let* $\mathcal{A}$ *be a timed automaton, and* $\equiv$ *denote any bisimilarity relation. A* ϵ-sampled implementation w.r.t. $\equiv$ *is a timed automaton* $\mathcal{A}'$ *such that*

(*i*) $\mathcal{A}'$ *is* ϵ*-samplable;*

(*ii*) $[\![\mathcal{A}']\!] \equiv [\![\mathcal{A}]\!]$.

Note that a similar phenomenon as in Fig. 10.2 does not occur in sampled semantics since sampling does not add extra behaviour, but may only remove some; so we do not need the ready simulation condition.

10.3.3 Constructions

We will present two constructions which yield an implementation for any timed automaton. In our first construction, for any timed automaton given with a safety specification, we construct a safety robust implementation. Our second construction is stronger: Given any timed automaton $\mathcal{A}$ and any desired $\epsilon > 0$, we construct a timed automaton $\mathcal{A}'$ which is both an ϵ-robust implementation and an ϵ-sampled implementation of $\mathcal{A}$ w.r.t. $\approx_{0^+}$ (we also give a variant w.r.t. $\sim_0$ for robustness).

Since, $\mathcal{A}$ and $\mathcal{A}'_\Delta$ are timed-action ϵ-bisimilar, all standard untimed linear- and branching-time properties (e.g. expressible in LTL, resp. CTL) proven for the original automaton are preserved in the implementation. An example of such a property is deadlock-freedom, which is an important

property of programs. Moreover, respective delays are close by ϵ. This also implies that in a quantitative extension of the computation tree logic studied in [FLT10], formulas approximately preserve their satisfaction values between two ϵ-bisimilar systems (see [BLM+11]).

Theorem 10.3.8. *Let $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E)$ be an integral timed automaton which is safe w.r.t. some set $\mathsf{Bad} \subseteq \Sigma$. Let W denote the number of regions of $\mathcal{A}$. Then,*

1. *There exists a safety robust implementation of $\mathcal{A}$ w.r.t $\sim_0$, with $|\mathcal{L}|$ locations, the same number of clocks and at most $|E| \cdot W$ edges.*
2. *For all $\epsilon > 0$, there exists a timed automaton $\mathcal{A}'$ which is a ϵ-robust implementation w.r.t. $\sim_0$; and a timed automaton $\mathcal{A}''$ which is both a ϵ-sampled and ϵ-robust implementation w.r.t. $\approx_{0^+}$. Both timed automata have the same number of clocks as $\mathcal{A}$, and the number of their locations and edges is bounded by $O(|\mathcal{L}| \cdot W \cdot (\frac{1}{\epsilon})^{|\mathcal{C}|})$.*

We will now detail the construtions stated in Theorem 10.3.8. The proofs are given in subsequent sections.

For any timed automaton $\mathcal{A}$ and any location ℓ of $\mathcal{A}$, let $\mathsf{Reach}(\llbracket\mathcal{A}\rrbracket)|_\ell$ denote the projection of the set of reachable states at location ℓ to $\mathbb{R}^{\mathcal{C}}_{\geq 0}$. For any ℓ, there exists guards $\phi^\ell_1, \ldots, \phi^\ell_{n_l}$ such that $\bigcup_i \llbracket\phi^\ell_i\rrbracket = \mathsf{Reach}(\llbracket\mathcal{A}\rrbracket)|_\ell$ (in fact, the set of reachable states at a given location is a union of regions but not necessarily convex). We use these formulas to construct a new automaton where we restrict all transitions to be activated only at reachable states.

Definition 10.3.9. *Let $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E)$ be any integral timed automaton. Define timed automaton $\mathsf{safe}(\mathcal{A})$ from $\mathcal{A}$ by replacing each edge $\ell \xrightarrow{\phi,\sigma,R} \ell'$, by edges $\ell \xrightarrow{\phi\wedge\phi^\ell_i,\sigma,R} \ell'$ for all $i \in \{1, \ldots, n_\ell\}$.*

As stated in Theorem 10.3.8 the worst-case complexity of this construction is exponential. However, in practice, $\mathsf{Reach}(\llbracket\mathcal{A}\rrbracket)|_\ell$ may have a simple shape, which can be captured by few formulas ϕ^ℓ_i.

Although the above construction will be enough to obtain a safety-robust timed automaton w.r.t. a given set Bad, it may not be bisimulation-robust. The following construction provides bisimulation-robustness.

Definition 10.3.10. *Let $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E)$ be an integral timed automaton. Let M be the largest constant that appears in $\mathcal{A}$, and let η be any granularity. We define $\mathsf{impl}_\eta(\mathcal{A})$ as a timed automaton over the set of locations ℓ_r where ℓ is a location of $\mathcal{A}$ and r is an (η, M)-region, and over the same set of clocks. Edges are defined as follows. Whenever there is an edge $\ell \xrightarrow{\phi,\sigma,R} \ell'$ in $\mathcal{A}$, we let $\ell_r \xrightarrow{\phi\wedge\phi_s,\sigma,R} \ell'_{s[R\leftarrow 0]}$, for all (η, M)-regions r and $s \in \mathsf{tsucc}^*(r)$ such that $\llbracket\phi_s\rrbracket \subseteq \llbracket\phi\rrbracket$.*

We define $\overline{\mathsf{impl}}_\eta(\mathcal{A})$ as the closed timed automaton obtained from $\mathsf{impl}_\eta(\mathcal{A})$ where each guard is replaced by its closed counterpart[1].

In this chapter, we always consider integral timed automata as input, and the only non-integer constants are those added by our construction. Observe that the size of $\mathsf{impl}_\eta(\mathcal{A})$ depends on η, since a smaller granularity yields a greater number of (η, M)-regions.

The main theorem is a direct corollary of the following lemma, where we state our results in detail. The bounds on the size of the constructed implementations follow by construction.

[1] that is, all $<$ are replaced by $\leq$, and $>$ by $\geq$.

Lemma 10.3.11. *Let $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E)$ be an integral timed automaton and fix any $\epsilon > 0$. Assume that $\mathcal{A}$ is safe w.r.t. some set $\mathsf{Bad} \subseteq \Sigma$. Then,*

1. *$\mathsf{safe}(\mathcal{A})$ is safety-robust, $[\![\mathcal{A}]\!] \sim_0 [\![\mathsf{safe}(\mathcal{A})]\!]$ and for any $\Delta < \frac{1}{2|\mathcal{C}|}$, $[\![\mathsf{safe}(\mathcal{A})_\Delta]\!] \sqsubseteq^{\mathsf{Bad}} [\![\mathcal{A}_\Delta]\!]$.*
2. *For any granularity η and $\Delta > 0$ such that $2(\eta + \Delta) < \epsilon$, we have $[\![\mathcal{A}]\!] \approx_{0+} [\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ and $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!] \approx_\epsilon [\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_\Delta]\!]$. Moreover, for any $0 < \Delta' < \Delta < \frac{1}{|\mathcal{C}|}$, $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_{\Delta'}]\!] \sqsubseteq^{\mathsf{Bad}} [\![\mathcal{A}_\Delta]\!]$.*
3. *For any granularity η and $\Delta > 0$ such that $2(\eta + \Delta) < \epsilon$, we have $[\![\mathcal{A}]\!] \sim_0 [\![\mathsf{impl}_\eta(\mathcal{A})]\!]$ and $[\![\mathsf{impl}_\eta(\mathcal{A})]\!] \approx_\epsilon [\![\mathsf{impl}_\eta(\mathcal{A})_\Delta]\!]$. Moreover, whenever $\Delta < \frac{1}{|\mathcal{C}|}$, $[\![\mathsf{impl}_\eta(\mathcal{A})_\Delta]\!] \sqsubseteq^{\mathsf{Bad}} [\![\mathcal{A}_\Delta]\!]$.*
4. *For any granularities η and α such that $\eta = k\alpha$ for some $k \in \mathbb{N}_{>0}$ and $\eta < \epsilon/2$, $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!] \approx_\epsilon [\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]^\alpha$.*

Note that both $\overline{\mathsf{impl}}_\eta(\mathcal{A})$ and $\mathsf{impl}_\eta(\mathcal{A})$ provide the relation $\approx_{\epsilon^+}$ between the specification (that is, $[\![\mathcal{A}]\!]$) and the implementation (that is, $[\![\mathcal{A}'_\Delta]\!]$). However, the latter has a stronger relation with $[\![\mathcal{A}]\!]$, so we also study it separately.

Trading precision against complexity. The choice of the granularity in $\overline{\mathsf{impl}}_\eta(\mathcal{A})$ and $\mathsf{impl}_\eta(\mathcal{A})$ allows one to obtain an implementation of $\mathcal{A}$ with any desired precision. However, this comes with a cost since the size of $\overline{\mathsf{impl}}_\eta(\mathcal{A})$ is exponential in the granularity η. But it is also possible to give up on precision in order to reduce the size of the implementation. In fact, one could define $\mathsf{impl}_\equiv(\mathcal{A})$ where the regions are replaced by the equivalence classes of any finite time-abstract bisimulation $\equiv$. Then, we get $[\![\mathcal{A}]\!] \approx_0 [\![\mathsf{impl}_\equiv(\mathcal{A})]\!]$ and $[\![\mathsf{impl}_\equiv(\mathcal{A})]\!]$ is time-abstract bisimilar to $[\![\mathsf{impl}_\equiv(\mathcal{A})_\Delta]\!]$ for any $\Delta > 0$. In order to obtain, say $[\![\mathsf{impl}_\equiv(\mathcal{A})]\!] \approx_K [\![\mathsf{impl}_\equiv(\mathcal{A})_\Delta]\!]$, for some desired $K \geq 1$, one could, roughly, split these bisimulation classes to sets of delay-width at most $O(K)$, that is the maximal delay within a bounded bisimulation class (there is a subtlety with unbounded classes, where, moreover, all states must have arbitrarily large time-successors within the class). Note however that safety specifications are only guaranteed to be preserved for small enough K (see Lemma 10.3.11).

10.4 Proof of Correctness

This section is devoted to the proof of Lemma 10.3.11. We start with general properties of regions, in subsection 10.4.1. In subsection 10.4.2, we prove the robustness of $\mathsf{impl}_\eta(\mathcal{A})$ and $\overline{\mathsf{impl}}_\eta(\mathcal{A})$, as stated in points 2 and 3 of Lemma 10.3.11. In subsection 10.4.3, we prove that $\overline{\mathsf{impl}}_\eta(\mathcal{A})$ is bisimulation-samplable (point 4). Last, safety preservation of all the above systems, including $\mathsf{safe}(\mathcal{A})$, is proved in subsection 10.4.4.

10.4.1 Properties of regions

We give several properties of the enlargement of regions. Fixing constants η and M, we refer to any (η, M)-region simply as a region.

Proposition 10.4.1. *Let $u \in \mathbb{R}^{\mathcal{C}}_{\geq 0}$ such that $u \in [\![\langle \phi_s \rangle_\Delta]\!]$ for some region s. Then for any subset of clocks $R \subseteq C$, $u[R \leftarrow 0] \in [\![\langle \phi_{s[R \leftarrow 0]} \rangle_\Delta]\!]$.*

The following proposition shows, intuitively, that enlarged guards cannot distinguish the points of an "enlarged region". The proof is straightforward using difference bound matrices in normal form. Note that the property does not hold if ϕ_s is not in normal form.

Proposition 10.4.2. *Let s denote a region, and ϕ a guard. If $[\![\phi_s]\!] \subseteq [\![\phi]\!]$, then $[\![\langle\phi_s\rangle_\Delta]\!] \subseteq [\![\langle\phi\rangle_\Delta]\!]$.*

Proposition 10.4.3. *Let $u \in \mathbb{R}^{\mathcal{C}}_{\geq 0}$ such that $u \in [\![\langle\phi_s\rangle_\Delta]\!]$ for some region s. Then for all $s' \in \mathsf{tsucc}^*(s)$, there exists $d \geq 0$ such that $u + d \in [\![\langle\phi_{s'}\rangle_\Delta]\!]$.*

Proof. We only need to prove the case where s' is the immediate time-successor of s. Let A denote the DBM that defines s, and A' the DBM that defines s'.

Let $u \in [\![\langle\phi_s\rangle_\Delta]\!]$. All time-successor regions of s contain the exact same diagonal constraints as s (since $(x+d)-(y+d) = x-y$.) Then all we need to show is that constraints of the form $-\mathsf{A}'_{0,x} - \Delta \prec^{A'}_{0,x} x \prec^{A'}_{x,0} \mathsf{A}'_{x,0} + \Delta$ is satisfied by a time-successor of u.

First, assume that some clocks have integer values in s, say $-A_{0,x} = A_{x,0}$ for $x \in I$ for some non-empty $I \subseteq \mathcal{C}$, and that $\mathsf{A}_{x,0} < M$ (the case where $\mathsf{A}_{x,0} = M$ is similar). Then A' is obtained by replacing these constraints by $\mathsf{A}_{x,0} < x < \mathsf{A}_{x,0} + 1$ for all $x \in I$. We have $\mathsf{A}_{x,0} - \Delta \leq u(x) \leq \mathsf{A}_{x,0} + \Delta$ for all $x \in I$, so the enlargement of the above constraint is satisfied by u unless $u(x) = \mathsf{A}_{x,0} - \Delta$ for some $x \in I$. In this case, we only need to delay $d > 0$ so that $\mathsf{A}_{x,0} - \Delta < u(x) + d < \mathsf{A}_{x,0}$. Since any constraint of A which is not an equality is a strict inequality, this delay can be chosen small enough so that all other constraints are still satisfied.

Now, suppose no clock has an integer value in s. Let x denote a clock that has the largest fractional part in s (there may be several clocks x). Then s satisfies $\mathsf{A}_{x,0} - 1 < x < \mathsf{A}_{x,0}$. But then A', which describes the immediate time-successor of s, is obtained from A by replacing the above constraint by $x = \mathsf{A}_{x,0}$. We know that $\mathsf{A}_{x,0} - 1 - \Delta < u(x) < \mathsf{A}_{x,0} + \Delta$ by hypothesis. If $u(x) \geq \mathsf{A}_{x,0} - \Delta$ then $u \in [\![\langle\phi_{s'}\rangle_\Delta]\!]$. Otherwise, we show that $u + d \in [\![\langle\phi_{s'}\rangle_\Delta]\!]$, where $d = \mathsf{A}_{x,0} - u(x)$. For all clocks y, we have to verify that $\mathsf{A}_{y,0} - 1 - \Delta < u(y) + d < \mathsf{A}_{y,0} + \Delta$ for all $x \neq y$ (these define A', along with the diagonal constraints which are the same as in A). We have,

$$\begin{aligned} & \mathsf{A}_{y,0} - 1 - \Delta < u(y) + (\mathsf{A}_{x,0} - u(x)) < \mathsf{A}_{y,0} + \Delta \\ \iff & \mathsf{A}_{y,0} - 1 - \mathsf{A}_{x,0} - \Delta < u(y) - u(x) < \mathsf{A}_{y,0} - \mathsf{A}_{x,0} + \Delta \\ \iff & \mathsf{A}_{x,0} - \mathsf{A}_{y,0} - \Delta < u(x) - u(y) < \mathsf{A}_{x,0} + 1 - \mathsf{A}_{y,0} + \Delta, \end{aligned}$$

which is true by the fact that ϕ_s is normalized. and that x has the greatest fractional part in s. □

The previous proposition is no longer valid if ϕ_s is not normalized. As an example, take the region defined by $x = 1 \wedge y = 0$, whose immediate successor is $1 < x < 2 \wedge 0 < y < 1 \wedge x - y = 1$. The enlargement of the former formula is satisfied by valuation $(x = 1 - \Delta, y = \Delta)$ but this has no time-successor that satisfies the enlargement of the latter.

Last, we need the following proposition which provides a bound on the delay that it takes to go from a region to another.

Proposition 10.4.4. *Let r be a region, and s a time-successor region of r, and $\Delta \geq 0$. Suppose that $u \in [\![\phi_r]\!]$ and $u + d \in [\![\phi_s]\!]$ for some $d \geq 0$. Then for any $v \in [\![\langle\phi_r\rangle_\Delta]\!]$, there exists $d' \geq 0$ such that $v + d' \in [\![\langle\phi_s\rangle_\Delta]\!]$ and $|d' - d| \leq 2\eta + 2\Delta$.*

Proof. First, observe that d' such that $v + d' \in [\![\langle\phi\rangle_\Delta]\!]$ exists by Proposition 10.4.3. We distinguish the following cases.

If all clocks are above the maximal constant M in r, then the statement is true for $d' = d$.

If all clocks are above the maximal constants only in s, then let x be the clock with the smallest value in r. Suppose that r satisfies $A_{x,0} - \eta < x < A_{x,0}$. Then d must be greater than $M - A_{x,0} - \Delta$,

whereas any $d' \geq M - A_{x,0} + \eta + \Delta$ suffices to have $v + d' \in s$ for any $v \in r$. Hence d' can be chosen so that $|d' - d| \leq \eta + 2\Delta$. The situation is similar if r satisfies $x = A_{x,0}$.

Suppose now that some clock x is below the maximal constant both in r and s. Suppose $A_{x,0} - \eta < x < A_{x,0}$ in r and $A'_{x,0} - \eta < x < A'_{x,0}$ in s, where $A'_{x,0} > A_{x,0}$. Then, for all $v \in r$ and $v + d' \in s$, d' satisfies $A'_{x,0} - A_{x,0} - \eta \leq d \leq A'_{x,0} - A_{x,0} + \eta$. And this also holds for u and $u + d$. Therefore we get $|d' - d| \leq 2\eta$. The remaining cases where x satisfies an equality in r or s are similar. □

10.4.2 Proof of Robustness

We first prove that $\mathsf{impl}_\eta(\mathcal{A})$ and $\overline{\mathsf{impl}}_\eta(\mathcal{A})$ are bisimulation-robust, for an appropriate ϵ, that is $[\![\mathcal{A}']\!] \approx_\epsilon [\![\mathcal{A}'_\Delta]\!]$ where $\mathcal{A}'$ denotes any of these (Lemma 10.4.5). Then we show "faithfulness" results: Lemma 10.4.7 shows that $[\![\mathcal{A}]\!] \sim_0 [\![\mathsf{impl}_\eta(\mathcal{A})]\!]$ and $[\![\mathsf{safe}(\mathcal{A})]\!] \sim_0 [\![\mathcal{A}]\!]$, and Lemma 10.4.8 shows that $[\![\mathcal{A}]\!] \approx_{0^+} [\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$.

Lemma 10.4.5. *For any timed automaton $\mathcal{A}$, any granularity η, and any $\Delta > 0$, we have $[\![\mathsf{impl}_\eta(\mathcal{A})]\!] \approx_{2\Delta+2\eta} [\![\mathsf{impl}_\eta(\mathcal{A})_\Delta]\!]$.*

Proof. We fix any η and Δ. We define relation $\mathcal{R} \subseteq (\mathcal{L} \times \mathbb{R}^C) \times (\mathcal{L} \times \mathbb{R}^C)$ between $[\![\mathsf{impl}_\eta(\mathcal{A})]\!]$ and $[\![\mathsf{impl}_\eta(\mathcal{A})_\Delta]\!]$ by $(\ell_r, u)\mathcal{R}(\ell'_{r'}, u')$ whenever $\ell_r = \ell'_{r'}$ and

$$\forall s \in \mathsf{tsucc}^*(r), \exists d \geq 0, u + d \in [\![\phi_s]\!] \iff \exists d' \geq 0, u' + d' \in [\![\langle \phi_s \rangle_\Delta]\!]. \tag{10.1}$$

Intuitively, relation $\mathcal{R}$ relates states which can reach, by a delay, the same set of regions: we require the first system to reach $[\![\phi_s]\!]$, while it is sufficient that the second one reaches $[\![\langle \phi_s \rangle_\Delta]\!]$, since its guards are enlarged by Δ.

Initial states satisfy $(\ell_{\mathbf{0}}, \mathbf{0})\mathcal{R}(\ell_{\mathbf{0}}, \mathbf{0})$. In fact all time-successor regions of $\mathbf{0}$ are reachable by a delay in both systems. Let (ℓ_r, u) and (ℓ_r, u') be two states of $[\![\mathsf{impl}_\eta(\mathcal{A})]\!]$ and $[\![\mathsf{impl}_\eta(\mathcal{A})_\Delta]\!]$ respectively, such that $(\ell_r, u)\mathcal{R}(\ell_r, u')$. Suppose (ℓ_r, u) delays d and takes the edge $\ell_r \xrightarrow{\phi \wedge \phi_s, \sigma, R} \ell'_{s[R \leftarrow 0]}$. By construction, s is a time-successor of r and $s \subseteq [\![\phi]\!]$. By (10.1), there exists $d' \geq 0$ such that $u' + d' \in [\![\langle \phi_s \rangle_\Delta]\!]$. Then we also have $u' + d' \in [\![\langle \phi \rangle_\Delta]\!]$ since $[\![\phi_s]\!] \subseteq [\![\phi]\!]$ by construction and by Proposition 10.4.2. It remains to show that $(\ell'_{s[R \leftarrow 0]}, (u + d)[R \leftarrow 0])\mathcal{R}(\ell'_{s[R \leftarrow 0]}, (u' + d')[R \leftarrow 0])$. We have $(u + d)[R \leftarrow 0] \in [\![\phi_{s[R \leftarrow 0]}]\!]$ and $(u' + d')[R \leftarrow 0] \in [\![\langle \phi_{s[R \leftarrow 0]} \rangle_\Delta]\!]$. The former follows from the properties of regions, and the latter from Proposition 10.4.1. Then the set of time successor regions that are reachable by a delay are exactly all time successor regions of $s[R \leftarrow 0]$ in both systems: in $[\![\mathsf{impl}_\eta(\mathcal{A})]\!]$ this is obvious from the region automaton construction, and in $[\![\mathsf{impl}_\eta(\mathcal{A})_\Delta]\!]$ from Proposition 10.4.3.

We now prove the symmetric case. Suppose (ℓ_r, u') delays d' and takes the edge $\ell_r \xrightarrow{\phi \wedge \phi_s, \sigma, R} \ell'_{s[R \leftarrow 0]}$. By construction, s is a time-successor of r and $s \subseteq [\![\phi]\!]$. By (10.1), there exists $d \geq 0$ such that $u + d \in [\![\phi_s]\!]$. Since $[\![\phi_s]\!] \subseteq [\![\phi]\!]$ by construction, $(\ell_r, u + d)$ satisfies the guard $\overline{\phi \wedge \phi_s}$. Now, one can show that $(\ell'_{s[R \leftarrow 0]}, (u + d)[R \leftarrow 0])\mathcal{R}(\ell'_{s[R \leftarrow 0]}, (u' + d')[R \leftarrow 0])$ as in the previous case.

The index $2(\eta + \Delta)$ of the bisimulation relation follows from Proposition 10.4.4. □

Lemma 10.4.6. *For any timed automaton $\mathcal{A}$, any granularity η, and any $\Delta > 0$, we have $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!] \approx_{2\Delta+2\eta} [\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_\Delta]\!]$.*

Proof. We fix any η and Δ, and define relation $R \subseteq (\mathcal{L} \times \mathbb{R}^C) \times (\mathcal{L} \times \mathbb{R}^C)$ between $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ and $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_\Delta]\!]$ by $(\ell_r, u)\mathcal{R}(\ell'_{r'}, u')$ whenever $\ell_r = \ell'_{r'}$ and

$$\forall s \in \mathsf{tsucc}^*(r), \exists d \geq 0, u + d \in [\![\overline{\phi_s}]\!] \iff \exists d' \geq 0, u' + d' \in [\![\overline{\langle \phi_s \rangle}_\Delta]\!]. \tag{10.2}$$

The difference with the previous lemma is that we now use the closures of formulas ϕ_r.

Initial states satisfy $(\ell_{\mathbf{0}}, \mathbf{0})\mathcal{R}(\ell_{\mathbf{0}}, \mathbf{0})$. In fact all closures of time-successor regions of $\mathbf{0}$ are reachable by a delay in both systems. Let (ℓ_r, u) and (ℓ_r, u') be two states of $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ and $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_\Delta]\!]$ respectively, such that $(\ell_r, u)\mathcal{R}(\ell_r, u')$. Suppose (ℓ_r, u) delays d and takes the edge $\ell_r \xrightarrow{\overline{\phi \wedge \phi_s}, a, R} \ell'_{s[R \leftarrow 0]}$. By construction, s is a time-successor of r and $s \subseteq [\![\phi]\!]$. By (10.2), there exists $d' \geq 0$ such that $u' + d' \in [\![\overline{\langle \phi_s \rangle}_\Delta]\!]$. Then we also have $u' + d' \in [\![\overline{\langle \phi \rangle}_\Delta]\!] = [\![\overline{\langle \phi \rangle_\Delta}]\!]$ since $[\![\phi_s]\!] \subseteq [\![\phi]\!]$ and by Proposition 10.4.2. It remains to show that $(\ell'_{s[R \leftarrow 0]}, (u+d)[R \leftarrow 0])\mathcal{R}(\ell'_{s[R \leftarrow 0]}, (u'+d')[R \leftarrow 0])$. We have $(u + d)[R \leftarrow 0] \in [\![\overline{\phi_{s[R \leftarrow 0]}}]\!]$ and $(u' + d')[R \leftarrow 0] \in [\![\overline{\langle \phi_{s[R \leftarrow 0]} \rangle}_\Delta]\!]$. The former follows from the properties of regions, and the latter from Proposition 10.4.1. Then the set of time successor regions that are reachable by a delay are exactly all time successor regions of $s[R \leftarrow 0]$ in both systems: in $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ this is obvious from the region automaton construction, and in $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_\Delta]\!]$ from Proposition 10.4.3.

We now prove the symmetric case although it is similar. Suppose (ℓ_r, u') delays d' and takes the edge $\ell_r \xrightarrow{\overline{\phi \wedge \phi_s}, a, R} \ell'_{s[R \leftarrow 0]}$. By construction, s is a time-successor of r and $s \subseteq [\![\phi]\!]$. By (10.2), there exists $d \geq 0$ such that $u + d \in [\![\overline{\phi_s}]\!]$. Since $[\![\phi_s]\!] \subseteq [\![\phi]\!]$ by construction, $(\ell_r, u + d)$ satisfies the guard $\phi \wedge \phi_s$. Now, one can show that $(\ell'_{s[R \leftarrow 0]}, (u + d)[R \leftarrow 0])\mathcal{R}(\ell'_{s[R \leftarrow 0]}, (u' + d')[R \leftarrow 0])$ as in the previous case.

The index $2(\eta + \Delta)$ of the bisimulation relation follows from Proposition 10.4.4. □

The parameter which we provide for the timed-action bisimilarity is (almost) tight. In fact, consider the automaton in Figure 10.2, where the guard of the edge entering ℓ_1 is changed to $x \leq 1$. Fix any η and Δ and consider the following cycle in $\overline{\mathsf{impl}}_\eta(\mathcal{A})$:

$$(\ell_{1,r_1}) \to (\ell_{2,r_2}) \to (\ell_{1,r_1}),$$

where r_1 is the region $1 - \eta < x < 1 \wedge y = 0$, and r_2 is the region $x = 0 \wedge 1 < y < 1 + \eta$. Suppose $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_\Delta]\!]$ first goes to location (ℓ_{1,r_1}) with $x = 1 + \Delta, y = 0$, and that this is matched in $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ by $(\ell_{1,r'_1}, (x = 1 - \alpha, y = 0))$ where necessarily $\alpha \geq 0$. It is shown in [DDMR08] that in any such cycle, the enlarged automaton can reach (by iterating the cycle) all states of the region r_1 at location ℓ_1 (See also Chapter 4). In particular, $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_\Delta]\!]$ can go to state $(\ell_{1,r_1}, (x = 1 - \eta, y = 0))$. However, without enlargement, all states $(\ell_1, r'_1, (v'_x, v'_y))$ reached from a state $(\ell_{1,r_1}, (v_x, v_y))$ with $v_y = 0$ satisfy $v'_x \geq v_x$, that is, the value of the clock x at location ℓ_1 cannot decrease along any run. Thus, the state $(\ell_{1,r_1}, (x = 1 - \eta, y = 0))$ of $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_\Delta]\!]$ is matched in $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ by some state $(\ell_{1,r'_1}, (v'_x, 0))$ where $v'_x \geq 1 - \alpha$. Now, from there, $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_\Delta]\!]$ can delay $1 + \Delta + \eta$ and go to ℓ_2, whereas $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ can delay at most $1 + \alpha$ to take the same transition. The difference between the delays at the first and the last step is then at least $\max\big(\Delta + \alpha, 1 + \Delta + \eta - (1 + \alpha)\big) \geq \Delta + \eta/2$.

Next, we show that $\mathsf{safe}(\mathcal{A})$ and $\mathsf{impl}_\eta(\mathcal{A})$ are strongly 0-bisimilar to $\mathcal{A}$.

Lemma 10.4.7. *For any timed automaton $\mathcal{A}$, we have $[\![\mathsf{safe}(\mathcal{A})]\!] \sim_0 [\![\mathcal{A}]\!]$, and $[\![\mathcal{A}]\!] \sim_0 [\![\mathsf{impl}_\eta(\mathcal{A})]\!]$ for any granularity η.* □

Proof. Fix any η. All regions in this proof will refer to (η, M)-regions where M is the maximal constant of $\mathcal{A}$. Let $[\![\mathcal{A}]\!] = (S, s_0, \Sigma, T)$ and $[\![\mathsf{impl}_\eta(\mathcal{A})]\!] = (S', s'_0, \Sigma, T')$, consider relation $\mathcal{R} \subseteq S \times S'$ defined by,

$$\mathcal{R} = \{(\ell, v), (\ell_r, v) \mid \mathsf{reg}(v) \in \mathsf{tsucc}^*(r), (\ell, v) \in S \text{ and } (\ell_r, v) \in S'\}.$$

Consider any $(\ell, v)\ \mathcal{R}\ (\ell_r, v)$ and suppose $(\ell, v) \xrightarrow{a} (\ell', v')$ in $[\![\mathcal{A}]\!]$. Then there must be an edge $\ell \xrightarrow{\phi,a,R} \ell'$ in $\mathcal{A}$ such that $v \in [\![\phi]\!]$ and $v' = v[R \leftarrow 0]$. By construction there exists an edge $\ell_r \xrightarrow{\phi \wedge \phi_{\mathsf{reg}(v)},a,R} \ell_{\mathsf{reg}(v)[R\leftarrow 0]}$ in $\mathsf{impl}_\eta(\mathcal{A})$. This is because $\mathsf{reg}(v) \in \mathsf{tsucc}^*(r)$ and $\mathsf{reg}(v) \subseteq [\![\phi]\!]$ (since $v \in [\![\phi]\!]$ and clock formulae cannot distinguish points in the same region). Valuation v obviously satisfies $\phi \wedge \phi_{\mathsf{reg}(v)}$. We have $v' = v[R \leftarrow 0]$ so $\mathsf{reg}(v') = \mathsf{reg}(v)[R \leftarrow 0]$, thus $\mathsf{reg}(v') \in \mathsf{tsucc}^*(\mathsf{reg}(v)[R \leftarrow 0])$. Therefore $(\ell', v')\mathcal{R}(\ell'_{\mathsf{reg}(v)[R\leftarrow 0]}, v')$.

The symmetric case, and the timed transitions are equally simple to see.

The proof of $[\![\mathcal{A}]\!] \sim_0 [\![\mathsf{safe}(\mathcal{A})]\!]$ follows from the fact that strengthening the guards by the description of reachable states does not change the semantics (it only does so for the enlarged automaton). □

The proof of $[\![\mathcal{A}]\!] \approx_{0^+} [\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ is trickier. In fact, since all guards are closed in $\overline{\mathsf{impl}}_\eta(\mathcal{A})$, but not necessarily in $\mathcal{A}$, all time delays may not be matched exactly. The first part of the proof follows the lines of Proposition 16 of [OW03a], who, by a similar construction, prove that the finite timed traces of $[\![\mathcal{A}]\!]$ are dense in those of $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$, for an appropriate topology. Their result has a similar flavor, but we consider 0^+-bisimulation which cannot be interpreted in terms of density in an obvious way.

Lemma 10.4.8. *For any timed automaton $\mathcal{A}$ and granularity η, $[\![\mathcal{A}]\!] \approx_{0^+} [\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$.*

Proof. We fix any η and $\delta \in (0, 1)$. We define $(\ell, v)\mathcal{R}(\ell_r, v')$ iff

$$r = \mathsf{reg}(v), v' \in \overline{\mathsf{reg}(v)} \text{ and } \exists v'' \in \mathsf{reg}(v), v = \delta v'' + (1 - \delta)v'. \tag{10.3}$$

We show that $\mathcal{R}$ is a timed-action 0^+-bisimulation.

Initial states clearly satisfy (10.3) by letting v'' be equal to both valuations. Consider now states $(\ell, v)\ \mathcal{R}\ (\ell_r, v')$. Let $v'' \in \mathsf{reg}(v)$ such that $v = \delta v'' + (1 - \delta)v'$. We prove the two directions.

- Suppose $(\ell_r, v') \xrightarrow{d',\sigma} (\ell'_{r'}, v' + d'[R \leftarrow 0])$ in $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$, using the edge $\ell_r \xrightarrow{\overline{\phi \wedge \phi_s},\sigma,R} \ell'_r$. Since s (the region described by ϕ_s) is a time-successor of $\mathsf{reg}(v)$ by construction, there exists some $d'' \geq 0$ such that $v'' + d'' \in s$.
  Then $(\ell, v) \xrightarrow{d,\sigma} (\ell', (v + d)[R \leftarrow 0])$ as

  $$(\ell, v) \xrightarrow{d} (\ell, v + d) \xrightarrow{\sigma} (\ell', (v + d)[R \leftarrow 0]),$$

  using the edge $l \xrightarrow{\phi,\sigma,R} \ell'$ in $\mathcal{A}$ and $d = \delta d'' + (1 - \delta)d'$. Now $v + d \in [\![\phi]\!]$ because $v'' + d'' \in s$, $v' + d' \in \overline{s}$ and $v + d = \delta(v'' + d'') + (1 - \delta)(v' + d')$ with $0 < \delta < 1$ (in fact, $v + d$ must be in $\mathsf{reg}(s)$ and not only in $\overline{\mathsf{reg}(s)}$).
  Also $(v + d)[R \leftarrow 0] = \delta(v'' + d'')[R \leftarrow 0] + (1 - \delta)(v' + d')[R \leftarrow 0]$, and $(v'' + d'')[R \leftarrow 0] \in s[R \leftarrow 0] = \mathsf{reg}(v + d)[R \leftarrow 0]$, thus $(\ell', (v + d)[R \leftarrow 0])\ \mathcal{R}\ (\ell'_{s[R\leftarrow 0]}, (v' + d')[R \leftarrow 0])$.

- Symmetrically, suppose that $(\ell, v) \xrightarrow{d,\sigma} (\ell', (v+d)[R \leftarrow 0])$ in $[\![\mathcal{A}]\!]$, using the edge $\ell \xrightarrow{\phi,\sigma,R} \ell'$. Let $s = \mathsf{reg}(v+d)$, then (by Lemma 10.4.9 below) there exists $d', d'' \geq 0$ such that $v' + d' \in \overline{s}$ and $v'' + d'' \in s$. Hence $v + d = \delta(v'' + d'') + (1-\delta)(v' + d')$.

  By construction $\overline{\mathsf{impl}}_\eta(\mathcal{A})$ contains $\ell_r \xrightarrow{\overline{\phi \wedge \phi_s},\sigma,R} \ell'_{s[R\leftarrow 0]}$ since $[\![\phi_s]\!] \subseteq [\![\phi]\!]$. Then we have $(\ell_r, v') \xrightarrow{d',\sigma} ({\ell'}_{s[R\leftarrow 0]}, (v' + d')[R \leftarrow 0])$. Observing $(v+d)[R \leftarrow 0] = \delta(v'' + d'')[R \leftarrow 0] + (1-\delta)(v' + d')[R \leftarrow 0]$, and $(v'' + d'')[R \leftarrow 0] \in s[R \leftarrow 0] = \mathsf{reg}(v+d)[R \leftarrow 0]$, we have $(\ell', (v+d)[R \leftarrow 0]) \mathcal{R} ({\ell'}_{s[R\leftarrow 0]}, (v' + d')[R \leftarrow 0])$.

We now prove that relation $\mathcal{R}$ has parameter at most 2δ. In fact, whenever we have $(\ell, v) \mathcal{R} (\ell_r, v')$, valuation v' and the corresponding valuation v'' reside in the (closure of the) same region, and at each transition, these delay into the (closure of the) same region. Then the difference between these delays is at most 2 (in fact, 2η) by Proposition 10.4.4. With the notations of the first case, we get $|d'' - d'| \leq 2$. But since $d = \delta d'' + (1-\delta)d'$, we have $|d - d'| = |\delta d'' - \delta d'| = \delta |d'' - d'| \leq 2\delta$.

The result follows since δ was chosen arbitrarily. □

Note that it may be the case that $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!] \not\sim_k [\![\mathcal{A}]\!]$ for all $k \geq 0$. In fact, consider a location with outgoing edges respectively guarded by $x < 1$, $x = 1$, $x > 1$. Then $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ is able to delay to $x = 1$ (say, starting from $x = 0$) where it satisfies all the guards but $[\![\mathcal{A}]\!]$ cannot delay to such a point.

Lemma 10.4.9. *Let $v, v', v'' \in \mathbb{R}_{\geq 0}$ such that $v'' \in \mathsf{reg}(v)$ and $v' \in \overline{\mathsf{reg}(v)}$, and $v = \epsilon v'' + (1-\epsilon)v'$ for some $\epsilon \in (0,1)$. Then for all $d \geq 0$, there exists $d', d'' \geq 0$ s.t. $v + d = \epsilon(v'' + d'') + (1-\epsilon)(v' + d')$, $v'' + d'' \in \mathsf{reg}(v+d)$ and $v' + d' \in \overline{\mathsf{reg}(v+d)}$.*

Proof. Write R_1 for region $\mathsf{reg}(v)$ and R_2 for its immediate successor. We will prove the property when $v + d \in R_2$. The general case is then a consequence. [The case where R_1 has no immediate successor is trivial.]

Let $D = \{v + d \mid d \in \mathbb{R}_{\geq 0}\}$, $D' = \{v' + d' \mid d' \in \mathbb{R}_{\geq 0}\}$ and $D'' = \{v'' + d'' \mid d'' \in \mathbb{R}_{\geq 0}\}$. We distinguish between two cases.

First case: assume $D \cap R_2$ is the singleton $\{v + d\}$. One of the constraints which define R_2 is of the form $x = c$. In particular, $D' \cap \overline{R_2}$ and $D'' \cap R_2$ are also singletons $\{v' + d'\}$ and $\{v'' + d''\}$ respectively. We then have $c = v(x) + d = v'(x) + d' = v''(x) + d''$, from which we deduce $d' = \epsilon(v''(x) - v'(x)) + d$ and $d'' = (1-\epsilon)(v'(x) - v''(x)) + d$. This is then a simple computation to get the desired property that $\epsilon(v'' + d'') + (1-\epsilon)(v' + d') = v + d$.

Second case: assume $D \cap R_2$ is not a singleton. In that case $D \cap R_1$ is the singleton $\{v\}$, and one of the constraints which define R_1 is of the form $x = c$. And $D' \cap \overline{R_1}$ and $D'' \cap R_1$ are also singletons, $\{v'\}$ and $\{v''\}$ respectively. If R_2 has no immediate successor, then this is easy, for any $d' > 0$ we can find $d'' > 0$ that satisfies the desired property. Assume that R_2 has an immediate successor R_3, one of the constraints that defines R_3 being $y = e$. Also, by definition of the immediate successor, for any $0 < d'$ such that $v'(y) + d' < e$, it is the case that $v' + d' \in \overline{R_2}$. Similarly for any $0 < d''$ such that $v''(y) + d'' < e$, it is the case that $v'' + d'' \in R_2$. Define $\lambda = \frac{d}{e - v(y)}$, and set $d' = \lambda(e - v'(y))$ and $d'' = \lambda(e - v''(y))$. We have that $v' + d' \in \overline{R_2}$ and $v'' + d'' \in R_2$. A simple computation yields that $\epsilon(v'' + d'') + (1-\epsilon)(v' + d') = v + d$.

This concludes the proof. □

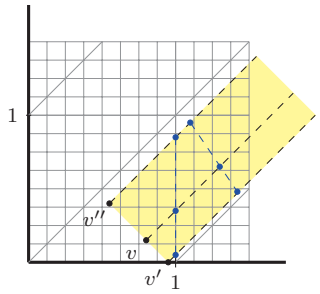


Figure 10.4: Lemma 10.4.9.

10.4.3 Proof of Samplability

We now show that $\overline{\mathsf{impl}}_\eta(\mathcal{A})$ is a sampled implementation for any timed automaton $\mathcal{A}$. This result follows from the following lemma and Lemma 10.4.8.

Lemma 10.4.10. *Let $\mathcal{A}$ be any integral timed automaton. For any granularities η and α such that $\eta = k\alpha$ for some $k \in \mathbb{N}_{>0}$, we have $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!] \approx_{2\eta} [\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]^\alpha$.*

The proof is given below and is similar to Lemma 10.4.5. But we first need the following lemma.

Let us fix some granularity η. In what follows, all regions refer to η-regions. We fix another granularity α such that $\eta = k\alpha$ for some $k \in \mathbb{N}_{>0}$.

Lemma 10.4.11. *Let r, s be regions such that $s \in \mathsf{tsucc}^*(r)$. For all $v \in \overline{r} \cap \mathbb{N}^{\mathcal{C}}$, there exists $d \in \alpha\mathbb{N}^{\mathcal{C}}$ such that $v + d \in \overline{s}$.*

Proof. The case where $r = s$ is trivial with $d = 0$. It is also easy if all clocks are above the maximal constant in s. Otherwise, we prove the lemma for the case where s is the immediate time successor of r.

In this proof only, an η-integer value means an element of $\eta\mathbb{N}$. Suppose that no clock has an η-integer value in r and let x denote the clock that has the greatest fractional part in r. Let $A_{x,0} - \eta < x < A_{x,0}$ denote the constraint on x in r. Then s is obtained from r by replacing this constraint by $x = A_{x,0}$ and without changing other constraints. We have $d = A_{x,0} - v(x) \in \alpha\mathbb{N}$ since $A_{x,0} \in \eta\mathbb{N} \subseteq \alpha\mathbb{N}$ and $v(x) \in \alpha\mathbb{N}$. We get $v + d \in \overline{s}$.

Suppose now that some subset of clocks $X \subseteq \mathcal{C}$ have η-integer values in r, say $x = A_{x,0}$ for each $x \in X$. Then the immediate time successor of r is obtained by replacing these equalities either by $A_{x,0} < x < A_{x,0} + \eta$ if $A_{x,0} < M$, or by $A_{x,0} > M$, without changing other contraints. We choose $d = 0$ and we have $v \in \overline{s}$. □

Proof of Lemma 10.4.10. We define the relation $R \subseteq (\mathcal{L} \times \mathbb{R}^C) \times (\mathcal{L} \times \alpha\mathbb{N}^C)$ between $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ and $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]^\alpha$ by $(\ell_r, u)\ \mathcal{R}\ (\ell'_{r'}, u')$ if, and only if $\ell_r = \ell'_{r'}$ and

$$\forall s \in \mathsf{tsucc}^*(r), \exists d \in \mathbb{R}_{\geq 0}, u + d \in [\![\overline{\phi_s}]\!] \iff \exists d' \in \alpha\mathbb{N}, u' + d' \in [\![\overline{\phi_s}]\!] \tag{10.4}$$

Initial states satisfy $(\ell_{\mathbf{0}}, \mathbf{0})\ \mathcal{R}\ (\ell_{\mathbf{0}}, \mathbf{0})$. In fact (the closures of) all time-successor regions of $\mathbf{0}$ are reachable by a delay in both systems (Lemma 10.4.11). Let (ℓ_r, u) and (ℓ_r, u') be two states of $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ and $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]^\alpha$ respectively, such that $(\ell_r, u)\ \mathcal{R}\ (\ell_r, u')$. Suppose (ℓ_r, u) delays d and

takes the edge $\ell_r \xrightarrow{\overline{\phi\wedge\phi_s},a,R} \ell'_{s[R\leftarrow 0]}$. By construction, s is a time-successor of r (or itself) and $s \subseteq [\![\phi]\!]$. By (10.4), there exists $d' \in \alpha\mathbb{N}$ such that $u'+d' \in [\![\overline{\phi_s}]\!]$. Then we also have $u'+d' \in [\![\overline{\phi}]\!]$ since $[\![\phi_s]\!] \subseteq [\![\phi]\!]$. It remains to show that $(\ell'_{s[R\leftarrow 0]}, (u+d)[R\leftarrow 0]) \mathcal{R} (\ell'_{s[R\leftarrow 0]}, (u'+d')[R\leftarrow 0])$. First, notice that we have $(u+d)[R\leftarrow 0] \in [\![\overline{\phi_{s[R\leftarrow 0]}}]\!]$ and $(u'+d')[R\leftarrow 0] \in [\![\overline{\phi_{s[R\leftarrow 0]}}]\!]$. Then both states can reach the closures of all time successors of $s[R\leftarrow 0]$: In $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ this is clear from the properties of regions and in $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]^\alpha$ from Lemma 10.4.11.

The symmetric case is similar. The index 2η follows from Proposition 10.4.4. □

10.4.4 Proof of Safety Preservation (Ready Simulation)

To complete the proof of Lemma 10.3.11, it remains to prove the ready simulation relations. The following proves the statement on $\mathsf{impl}_\eta(\mathcal{A})$ and $\overline{\mathsf{impl}}_\eta(\mathcal{A})$ of Lemma 10.4.12. The ready-simulation involving $\mathsf{safe}(\mathcal{A})$ is restated and proved afterwards.

Lemma 10.4.12. *We have* $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_{\Delta'}]\!] \sqsubseteq^{\mathsf{Bad}} [\![\mathcal{A}_\Delta]\!]$ *and* $[\![\mathsf{impl}_\eta(\mathcal{A})_\Delta]\!] \sqsubseteq^{\mathsf{Bad}} [\![\mathcal{A}_\Delta]\!]$ *for any* $0 < \Delta' < \Delta < \frac{1}{|\mathcal{C}|}$*; and* $[\![\mathsf{safe}(\mathcal{A})_\Delta]\!] \sqsubseteq^{\mathsf{Bad}} [\![\mathcal{A}_\Delta]\!]$ *for any* $\Delta < \frac{1}{2|\mathcal{C}|}$.

Proof. We prove the lemma for $\overline{\mathsf{impl}}_\eta(\mathcal{A})_\Delta$. The case of $\mathsf{impl}_\eta(\mathcal{A})_\Delta$ is similar (choosing $\Delta' = \Delta$).

We fix any η. Let $[\![\mathcal{A}_\Delta]\!] = (S, s_0, \Sigma, T$ and $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_{\Delta'}]\!] = (S', s'_0, \Sigma, T')$ respective TTSs. Let $\mathcal{R} \subseteq S \times S'$ defined by, $(\ell_r, u)\ \mathcal{R}\ (\ell, u)$ iff (ℓ_r, u) is reachable and $u \in [\![\overline{\langle\phi_r\rangle}_{\Delta'}]\!]$. We first show that $\mathcal{R}$ is a simulation with parameter 0, then prove that it is a ready simulation w.r.t. Bad.

Initial states are obviously related by $\mathcal{R}$. Now, consider any $(\ell_r, u)\mathcal{R}(\ell, u)$ and suppose that $(\ell_r, u) \xrightarrow{d,\sigma} (\ell'_{r'}, u')$ in $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_{\Delta'}]\!]$, for some $d \geq 0$ and $\sigma \in \Sigma$ following edge $\ell_r \xrightarrow{\sigma,\overline{\phi\wedge\phi_s},R} \ell'_{r'}$, where $s \in \mathsf{tsucc}^*(r)$. Then, $\mathcal{A}$ has the edge $\ell \xrightarrow{\sigma,\phi,R} \ell'$. Since $[\![\overline{\langle\phi\rangle}_{\Delta'}]\!] \subseteq [\![\langle\phi\rangle_\Delta]\!]$ (because $\Delta' < \Delta$) we have $u + d \in [\![\langle\phi\rangle_\Delta]\!]$, and we get $(\ell, u) \xrightarrow{d,\sigma} (\ell', u')$ in $[\![\mathcal{A}_\Delta]\!]$. Clearly, $(\ell'_{r'}, u')\ \mathcal{R}\ (\ell', u')$.

We now show that actions Bad are not enabled in $[\![\mathcal{A}_\Delta]\!]$ at any point in this simulation, which suffices to show ready simulation. First, observe that if any location ℓ_r is reachable in $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_{\Delta'}]\!]$, then all points of region r at location ℓ are reachable in $[\![\mathcal{A}]\!]$. This follows from the fact that $\overline{\mathsf{impl}}_\eta(\mathcal{A})$ encodes the region automaton in its locations, so any run of $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})_{\Delta'}]\!]$ follows the run of the region automaton of $\mathcal{A}$, made of the sequence of regions annotating each location. Now, suppose that $(\ell_r, u)\mathcal{R}(\ell, u)$ and $(\ell, u) \xrightarrow{d,\sigma} (\ell', u')$ in $[\![\mathcal{A}_\Delta]\!]$ for some $d \geq 0$ and $\sigma \in \mathsf{Bad}$. Let ϕ be the guard of this edge labelled by σ, and let $s = \mathsf{reg}(u+d)$. Since $u + d \in [\![\langle\phi\rangle_\Delta]\!]$, it must be the case that $d_\infty(s, [\![\phi]\!]) \leq \Delta$. But since $\Delta < \frac{\eta}{|\mathcal{C}|}$ by assumption, we have $\overline{s} \cap [\![\phi]\!] \neq \emptyset$ by Proposition 10.4.13, hence $d_\infty(s, [\![\phi]\!]) = 0$. This implies that $\overline{s}$ (thus, $\overline{r}$) is not reachable in $[\![\overline{\mathsf{impl}}_\eta(\mathcal{A})]\!]$ since otherwise, $\mathcal{A}$ would not be safe w.r.t. Bad. This shows that actions in Bad are not enabled in this simulation starting from the initial states. □

Proposition 10.4.13 ([DDMR08])**.** *Let R and R' be closures of η-regions of $\mathbb{R}_{\geq 0}$. If $R \cap R' = \emptyset$ then $d(R, R') \geq \frac{\eta}{|\mathcal{C}|}$.*

Lemma 10.4.14. *We have* $[\![\mathsf{safe}(\mathcal{A})_\Delta]\!] \sqsubseteq^{\mathsf{Bad}} [\![\mathcal{A}_\Delta]\!]$ *for any* $\Delta < \frac{1}{2|\mathcal{C}|}$.

Proof. The identity relation, restricted to the set of reachable states, is clearly a simulation. To see that it is also a ready simulation, consider any reachable state (ℓ, u) of $[\![\mathsf{safe}(\mathcal{A})_\Delta]\!]$, and suppose

that from this state, an action of Bad is enabled in $[\![\mathcal{A}_\Delta]\!]$, after a possible delay. There exists $d \geq 0$ such that $u + d \in [\![\langle\phi\rangle_\Delta]\!]$, where ϕ denotes the guard of the edge with a label in Bad. State (ℓ, u) cannot be the initial state since this would contradict the safety of $[\![\mathcal{A}]\!]$. In fact, we would have $d_\infty(\mathsf{reg}(\mathbf{d}), [\![\phi]\!]) \leq \Delta$ which implies $[\![\phi]\!] \cap \mathsf{reg}(\mathbf{d}) \neq \emptyset$. Therefore, (ℓ, u) must have a reachable predecessor in $[\![\mathsf{safe}(\mathcal{A})_\Delta]\!]$, say $(\ell', u') \xrightarrow{d',\sigma'} (\ell, u)$, following some edge $\ell' \xrightarrow{\sigma', g \wedge \phi_j^\ell, R} \ell$. By definition, all points of $[\![\phi_j^\ell]\!]$ are reachable in $[\![\mathcal{A}]\!]$ at location ℓ. We have $u' + d' \in [\![\langle\phi_j^\ell\rangle_\Delta]\!]$. So there exists a state (ℓ, v) which is reachable in $[\![\mathcal{A}]\!]$ such that $d_\infty(u, v) \leq \Delta$. But, we assumed that $u + d \in [\![\langle\phi\rangle_\Delta]\!]$, so $d_\infty(v + d, u + d) \leq \Delta$ implies $d_\infty(v + d, [\![\phi]\!]) \leq 2\Delta$. So $d_\infty(\mathsf{reg}(v + d), [\![\phi]\!]) = 0$ by Proposition 10.4.13, where $\mathsf{reg}(v + d)$ is reachable in $[\![\mathcal{A}]\!]$. We get $d_\infty(\mathsf{Reach}([\![\mathcal{A}]\!]), \{\ell\} \times [\![\phi]\!]) = 0$, which is a contradiction. □

10.5 Application to Robust Undecidability

The constructions we gave in this chapter are also important theoretical results, since they show that the class of robust timed automata contain timed automata that are equivalent, in the sense of strong 0-bisimulation, to arbitrary timed automata. Hence, considering only robust timed automata in the domain of any problem is not restrictive at all, when it comes to decidability and expressiveness. Although there is a blow-up in the size of the equivalent models we define, this is irrelevant if we are only interested in the decidability of verification problems.

Here, we apply the implementation constructions given previously to show that all undecidable problems on timed automata remain undecidable on the class of robust timed automata. We state explicitly the undecidability of the timed language universality, which is one of the important problems in verification.

The *timed language* of a timed automaton is the set of its timed traces of its initialized runs. The *timed language universality* problem asks whether the timed language of a given timed automaton contains all timed traces. The problem is undecidable for general timed automata [AD94].

Theorem 10.5.1. *The timed language universality problem is undecidable for the class of safety-robust (resp. bisimulation-robust) timed automata.*

Proof. The proof given in [AD94] consists in defining a language L_{undec} encoding the computations of a Minsky machine. Then a timed automaton $\mathcal{A}$ that encodes exactly the complement of L_{undec} is given. Hence, the universality of the timed language of $\mathcal{A}$ is equivalent to the non-emptiness of the computations of the Minsky machine. In this proof, we can replace $\mathcal{A}$ by $\mathsf{safe}(\mathcal{A})$ which has the same timed language since $\mathcal{A} \sim_0 \mathsf{safe}(\mathcal{A})$. Furthermore, $\mathsf{safe}(\mathcal{A})$ is safety-robust by construction. The same holds for $\mathsf{impl}_\eta(\mathcal{A})$. Thus, the problem remains undecidable in both classes of robust timed automata. □

The proof above shows that any encoding of an undecidable language by a regular timed language, can be realized by a robust timed automaton. One can thus formulate the following meta-theorem:

> *All problems whose undecidability is proven by encoding an undecidable problem as a timed automaton remain undecidable for the class of robust timed automata.*

10.6 Conclusion

We have presented a way to transform any timed automaton into robust and samplable ones, while preserving the original semantics with any desired precision. Such a transformation is interesting if the timed automaton under study is not robust (or not samplable), or cannot be certified as such. In this case, one can simply model-check the original automaton in the exact semantics for desired properties, then apply our constructions. This allows one to design timed automata at a high abstraction level, without taking into account imprecisions, and then automatically refine to an equivalent but robust design.

Our results also show that robust timed automata are not less expressive than the class of arbitrary timed automata, since one can always transform a timed automaton into a robust one and preserve its semantics exactly. On one hand, this is good news since one can *implement* all timed automata (satisfying the safety assumption of Definition 10.3.2). On the other hand, this means that our notion of robustness does not rule out very complex timed automata, such as those encoding undecidable problems: as we showed in Section 10.5, undecidable problems remain undecidable for the class of robust timed automata.

Several important questions remain open. First of all, for now, our construction is applied to timed automata seen as a whole. Thus, if the input is a network of timed automata, the construction requires each component to have (read-only) access to all other clocks. If all clocks cannot be publicly read, then our constructions may not be actually realizable in some platforms. Another problem is that we can only treat "one-player timed automata", that is, we assume that the evolution of the system is entirely controlled by the controller. In reality, there may be uncontrollable actions, whose guards cannot be constrained. Taking into account uncontrollable transitions would require formalizing the system as a game. Another interesting future work would be to investigate the *size complexity* of robust timed automata: given a timed automaton $\mathcal{A}$, can we always find a robust equivalent timed automaton of size polynomial in $\mathcal{A}$? This would not only extend our understanding of the class of robust timed automata by providing information on their succinctness, but it would also have practical implications since one would avoid the exponential blow-up of our constructions.

Chapter 11

Implementation by Shrinking

11.1 Introduction

The perturbation models and the corresponding robustness problems considered in this thesis are related to *implementability*, *i.e.*, whether a given model can be implemented on physical hardware, preserving (the properties of) its exact semantics. In this chapter, we first present a target *implementation semantics* for timed automata, which takes into account nonzero reaction times, synchronization delays and clock imprecisions. Our semantics is similar to the one studied in [DDR05a] with minor differences (corresponding to different abstraction choices), and we prove additional properties besides the one given there. We then show how shrinkability (of Chapter 5) can be used to ensure that the behavior is preserved in implementation.

We will "implement" timed automata simply by shrinking their guards. We explain the intuition in a simple case, where the only imprecisions are modelled as guard enlargement. When an edge with the guard $x \in [a, b]$ is implemented as $x \in [a+\delta, b-\delta]$, under imprecisions bounded by some Δ, the system still guarantees that $x \in [a+\delta-\Delta, b-\delta+\Delta]$. Provided that $0 \leq \Delta < \delta$, the resulting behavior then conforms to the initial model since

$$[a+\delta-\Delta, b-\delta+\Delta] \subseteq [a, b].$$

Thus, no new behavior is present in the implemented system under imprecisions. We will follow this idea to show that shrunk timed automata have no additional behavior in the implementation semantics than in the exact semantics. Furthermore, we are interested in ensuring that the implementation semantics also do not lose behaviors: We will show that if a timed automaton is shrinkable, then its implementation semantics is non-blocking and time-abstract simulates its exact semantics. Thus, the main result of this chapter is that shrinkable timed automata are implementable in our implementation semantics.

Implementability is in general a difficult problem, and an exact answer needs to take into account a detailed model of the platform, such as the worst and best case execution times for each instruction in a given microprocessor (*e.g.* [BKW12]). Such a modeling is out of the scope of this thesis. It is nonetheless useful to carry the formal verification as far as possible in the design of a system, so as to gain confidence in the design at hand. Our goal here is to show that some properties of the system (simulation and non-blockingness) are preserved in a semantics that is closer to a real implementation than the idealized abstract semantics of timed automata.

We first define our semantics and state its properties, then compare it with [DDR05a].

11.2 Implementation Semantics

In this chapter, we will adopt a different definition for timed transition systems, by making explicit the global time at any state. This will allow us to compare different measures of time between

different semantics. The time domain is the nonnegative reals. Given a TTS $(S, s_0, \Sigma, \mathbb{R}_{\geq 0}, \rightarrow)$, we do not separate delays and actions, but rather only define actions with *timestamps*. Hence, delays are now implicit. Transitions are labelled by $\sigma(T)$, with $T \in \mathbb{R}_{\geq 0}$ the *timestamp* of *action* $\sigma \in \Sigma$. In all TTSs we consider, the timestamps of consecutive actions are assumed to be nondecreasing. The definition of simulation can be adapted as follows. A relation $\mathcal{R} \subseteq S \times S$ is a simulation if for any $(s,t) \in \mathcal{R}$, whenever $s \xrightarrow{\sigma(T)} s'$, there exists t' such that $t \xrightarrow{\sigma(T)} t'$, for any $\sigma \in \Sigma$ and $T \in \mathbb{R}_{\geq 0}$. A timed automaton $\mathcal{A}$ is a *timed refinement* of $\mathcal{A}'$ if the initial state of $\mathcal{A}'$ simulates the initial state of $\mathcal{A}$. As in Chapter 5, we only consider closed timed automata.

We describe a system which interacts, via sending and receiving signals, with a physical environment (*e.g.* via sensors). We distinguish input and output actions, and define the transitions of the system taking into account the imprecisions of the clock, the transmission delay of signals and the reaction time of the system. When an event is generated at time T by the environment, it is treated by the system at time $T + \epsilon$, for some $\epsilon > 0$ which will be bounded but unpredictable. Similarly, when the environment receives a signal at time T, it must have been sent at some time $T - \epsilon$. We assume that the system ignores any signal that is received during the treatment of the previous signal; this *reaction time* will be also bounded but unpredictable. Thus, in our semantics, the system does not have a buffer to store incoming signals; it either responds immediately to a signal or ignores it. We define the timestamps of both input and output actions as the reaction times of the environment, since we are interested in the behavior of the environment controlled by a digital timed system.

The implementation semantics has three parameters:

a) Δ_c is the clock period,

b) Δ_r is the maximum *reaction time*, following each action,

c) Δ_t is the maximum *transmission delay* of signals between the system and the environment (ϵ above).

We suppose the system has a Δ_c-periodic clock, whose value, at any real time T, is $\lfloor T \rfloor_{\Delta_c} = \max_{k \in \mathbb{N}} \{k\Delta_c \mid k\Delta_c \leq T\}$.

Definition 11.2.1. *Let $\mathcal{A} = (\mathcal{L}, \ell_0, \mathcal{C}, \Sigma, E)$ be a timed automaton with $\Sigma = \Sigma_{in} \cup \Sigma_{out}$, and $\Delta_r, \Delta_c, \Delta_t > 0$. The* implementation semantics $[\![\mathcal{A}]\!]^{Impl}$ *is the TTS $(S_{\mathcal{A}}, s_0, \Sigma, E)$ in which states are tuples (ℓ, T, v, u_0): ℓ is a location, $T \in \mathbb{R}_{\geq 0}$ the current global time*[1]*, $v \in \mathbb{R}^{\mathcal{C}}_{\geq 0}$ the timestamp of the latest reset for each clock, and $u_0 \in [0, \Delta_r]$ the reaction time following the latest location change. From any state (ℓ, T, v, u_0), for any edge $\ell \xrightarrow{\sigma, g, R} \ell'$ and $T' \geq T$, we let,*

- *if $\sigma \in \Sigma_{in}$, $(\ell, T, v, u_0) \xrightarrow{\sigma(T')} (\ell', T' + \epsilon, v[R \leftarrow T' + \epsilon], u'_0)$, whenever $\lfloor T' + \epsilon \rfloor_{\Delta_c} - \lfloor v \rfloor_{\Delta_c} \models g$ and $T' + \epsilon \geq T + u_0$, where $(\epsilon, u'_0) \in [0, \Delta_t] \times [0, \Delta_r]$ is chosen non-deterministically,*

- *if $\sigma \in \Sigma_{out}$, $(\ell, T, v, u_0) \xrightarrow{\sigma(T')} (\ell', T', v[R \leftarrow T' - \epsilon], u'_0)$, whenever $\lfloor T' - \epsilon \rfloor_{\Delta_c} - \lfloor v \rfloor_{\Delta_c} \models g$, $\epsilon < (T' - T)$, and $T' - \epsilon \geq T + u_0$, where $(\epsilon, u'_0) \in [0, \Delta_t] \times [0, \Delta_r]$ is chosen non-deterministically.*

[1] Although we define the semantics with respect to an exact global time, the behavior of this TTS will only depend on an approximate measure of this time.

The transitions should be interpreted as follows. If the environment generates an event at time T', then the system responds to it at time $T' + \epsilon$, provided that the reaction time from the previous event is over ($T' + \epsilon \geq T + u_0$), and the guard is approximately satisfied. If the system generates an event at time $T' - \epsilon$, similar constraints apply but the timestamp is registered as T', which is the time the environment receives the event. Notice that ϵ and u_0 are bounded by known values but are unpredictable, so they cannot be chosen by the system. We will consider *scheduler* functions ρ, which, depending on the history of a given run, chooses (ϵ, u_0) at each transition. For any scheduler ρ, we denote by $[\![\mathcal{A}]\!]_\rho^{\mathsf{Impl}}$ the implementation semantics, where (ϵ, u_0) is given by ρ at each transition. We will not formally define ρ here, but it can be done without difficulty.

The following proposition states the relation between the exact semantics and the implementation semantics of timed automata. All properties hold under *any* scheduler ρ. For any TTS $\mathcal{T}$, let us write $\mathcal{T}^{\geq\alpha}$, the TTS obtained from $\mathcal{T}$ where consecutive transitions are separated by at least α time units.

Proposition 11.2.2. *Let $\mathcal{A}$ be a closed non-blocking timed automaton, and $\Delta_r, \Delta_c, \Delta_t > 0$. Then, for any $\Delta \geq 2\Delta_c + 4\Delta_t + \Delta_r$ and scheduler ρ, $[\![\mathcal{A}_\Delta]\!]_\rho^{\mathsf{Impl}}$ is non-blocking and,*

$$[\![\mathcal{A}]\!]^{\geq 2\Delta_r+\Delta_t} \sqsubseteq [\![\mathcal{A}_\Delta]\!]_\rho^{\mathsf{Impl}} \sqsubseteq [\![\mathcal{A}_{\Delta+2\Delta_c+4\Delta_t}]\!].$$

Before proving this proposition, let us give our main result on implementability. We assume that all timed automata contain an additional clock u that is reset at each edge, and that all edges are guarded by $u \geq 0$. Clearly, this does not change the semantics of timed automata, but we will require these guards to be shrunk. We have the following result.

Theorem 11.2.3. *Let $\mathcal{A}$ be a closed timed automaton that is strongly shrinkable w.r.t. $\mathcal{A}$ and let $\mathcal{A}_{-\mathbf{k}\delta}$ be its witnessing shrinking for $\delta \in [0, \delta_0]$. Let $\Delta_r, \Delta_c, \Delta_t > 0$ be parameters such that $4\Delta_c + 8\Delta_t + 2\Delta_r \leq \delta_0$. Then for all $\Delta \in [2\Delta_c + 4\Delta_t + \Delta_r, \delta_0 - 2\Delta_c - 4\Delta_t]$, for any scheduler ρ, $[\![\mathcal{A}_\Delta]\!]_\rho^{\mathsf{Impl}}$ is a non-blocking timed refinement of $\mathcal{A}$ and time-abstract simulates $\mathcal{A}$.*

Proof. By Proposition 11.2.2 applied to $\mathcal{A}_{-\mathbf{k}\delta}$, we have

$$[\![\mathcal{A}_{-\mathbf{k}\delta}]\!] \quad \sqsubseteq \quad [\![\mathcal{A}_{-\mathbf{k}\delta+\Delta}]\!]_\rho^{\mathsf{Impl}} \quad \sqsubseteq \quad [\![\mathcal{A}_{-\mathbf{k}\delta+\Delta'}]\!] \quad \sqsubseteq \quad [\![\mathcal{A}]\!],$$

and $[\![\mathcal{A}_{-\mathbf{k}\delta+\Delta}]\!]_\rho^{\mathsf{Impl}}$ is non-blocking whenever $\Delta \geq 2\Delta_c + 4\Delta_t + \Delta_r$, $\Delta' = \Delta + 2\Delta_c + 4\Delta_t$ and $\delta \geq \max(2\Delta_r + \Delta_t, \Delta')$. In fact, $[\![\mathcal{A}_{-\mathbf{k}\delta}]\!]^{\geq 2\Delta_r+\Delta_t}$ is equal to $[\![\mathcal{A}_{-\mathbf{k}\delta}]\!]$ whenever $\delta \geq 2\Delta_r + \Delta_t$ due to the shrinking of the additional clock constraints in $\mathcal{A}$. The rightmost simulation is due to the fact that $-\mathbf{k}\delta + \Delta' \leq 0$. □

Thus, given δ_0, the parameters $\Delta_c, \Delta_t, \Delta_r$ and Δ can be chosen so that the implementation semantics of the automaton $\mathcal{A}_{-\mathbf{k}\delta+\Delta}$ is a timed refinement of the exact semantics of the original automaton. Moreover, when $\mathcal{A}$ is shrinkable (say, with parameters $\mathbf{k}\delta$), then $[\![\mathcal{A}_{-\mathbf{k}\delta+\Delta}]\!]_\rho^{\mathsf{Impl}}$ is also non-blocking and $[\![\mathcal{A}]\!] \sqsubseteq_{\text{t.a.}} [\![\mathcal{A}_{-\mathbf{k}\delta+\Delta}]\!]_\rho^{\mathsf{Impl}}$. Thus, shrinkable timed automata can be implemented so as to preserve non-blockingness and the behavior upto time-abstract simulation.

11.3 Proof of Proposition 11.2.2

We prove Proposition 11.2.2 through Lemmas 11.3.1 – 11.3.5. In the proofs we use the standard supremum distance on $\mathbb{R}^n$ defined by $d_\infty(\nu, \nu') = \max_{1\leq i\leq n}(|\nu_i - \nu'_i|)$, where $\nu, \nu' \in \mathbb{R}^n$. For any vector ν and real α, we denote by $\nu + \alpha$ the vector obtained by adding α to all components of ν.

Lemma 11.3.1. *Let $\mathcal{A}$ be a closed timed automaton, $\Delta_r, \Delta_c, \Delta_t > 0$ denote the parameters. Then, for any $\Delta \geq 2\Delta_c + 4\Delta_t$, and any scheduler ρ,*

$$[\![\mathcal{A}]\!]_\rho^{\mathit{Impl}} \sqsubseteq [\![\mathcal{A}_\Delta]\!].$$

Proof. We show that the relation $\mathcal{R}$ defined by $(\ell, T, v, u_0)\mathcal{R}(\ell, \nu, T')$ such that $T' \in [T - \Delta_t, T]$, and $d_\infty(\nu + T - T', T - v) \leq \Delta_t$ is a timed simulation. Notice that we do not require these two states to be at the same time instant, but the difference between these instants must be bounded and the clock valuations must be close when the second system delays to time T. Consider such a pair of states.

Consider the following transition.

$$(\ell, T, v, u_0) \xrightarrow{\sigma(T+\tau)} (\ell', T + \tau + \epsilon, v[R \leftarrow T + \tau + \epsilon], u_0'),$$

for some $\sigma \in \Sigma_{\mathsf{in}}$, and $(u_0', \epsilon) \in [0, \Delta_r] \times [0, \Delta_t]$ is given by ρ, and $\lfloor T + \tau + \epsilon \rfloor_{\Delta_c} - \lfloor v \rfloor_{\Delta_c} \models g$. We show that the following transitions are realizable in $[\![\mathcal{A}_\Delta]\!]$.

$$(\ell, \nu, T') \xrightarrow{\sigma(T+\tau)} (\ell', (\nu + T + \tau - T')[R \leftarrow 0], T + \tau).$$

We need to show that $\nu + T + \tau - T' \models \langle g \rangle_{2\Delta_c + 4\Delta_t}$. Using the fact that $\lfloor \alpha \rfloor_{\Delta_c} \in [\alpha - \Delta_c, \alpha]$ for any $\alpha \in \mathbb{R}_{\geq 0}$, we get $d_\infty\big((T + \tau + \epsilon - v), (\lfloor T + \tau + \epsilon \rfloor_{\Delta_c} - \lfloor v \rfloor_{\Delta_c})\big) \leq \Delta_c$. Then $T + \tau + \epsilon - v \models \langle g \rangle_{2\Delta_c}$ (the factor 2 is due to diagonal constraints), and, $T + \tau - v \models \langle g \rangle_{2\Delta_c + 2\Delta_t}$. Since $d_\infty(\nu + T - T', T - v) \leq \Delta_t$, we also have $d_\infty(\nu + T - T' + \tau, T - v + \tau) \leq \Delta_t$, so $\nu + T + \tau - T' \models \langle g \rangle_{2\Delta_c + 4\Delta_t}$. We now show that the new states are related by $\mathcal{R}$. Let $\nu' = (\nu + T + \tau - T')[R \leftarrow 0]$ and $v' = v[R \leftarrow T + \tau + \epsilon]$. We have $d_\infty(\nu' + \epsilon, T + \tau + \epsilon - v') \leq \Delta_t$. In fact, for any clock $x \notin R$, this follows from the assumption that $d_\infty(\nu + T - T', T - v) \leq \Delta_t$, and for all $x \in R$, we have $\nu'(x) + \epsilon = \epsilon \leq \Delta_t$ and $(T + \tau + \epsilon - v')(x) = 0$.

Now, consider the following transition.

$$(\ell, T, v, u_0) \xrightarrow{\sigma(T+\tau)} (\ell', T + \tau, v[R \leftarrow T + \tau - \epsilon], \epsilon, u_0'),$$

where $\sigma \in \Sigma_{\mathsf{out}}$, $(u_0', \epsilon) \in [0, \Delta_r] \times [0, \Delta_t]$ is given by ρ and $\lfloor T + \tau - \epsilon \rfloor_{\Delta_c} - \lfloor v \rfloor_{\Delta_c} \models g$. We show that the following transition is realizable in $[\![\mathcal{A}_\Delta]\!]$.

$$(\ell, \nu, T') \xrightarrow{\sigma(T+\tau)} (\ell', (\nu + T + \tau - T')[R \leftarrow 0], T + \tau)$$

We show that $\nu + T + \tau - T' \models \langle g \rangle_{2\Delta_c + 4\Delta_t}$. As in the previous case, we have, $T + \tau - \epsilon - v \models \langle g \rangle_{2\Delta_c}$ and using the fact that $d_\infty(\nu + T - T', T - v) \leq \Delta_t$, we get $\nu + T + \tau - T' \models \langle g \rangle_{2\Delta_c + 4\Delta_t}$. Let $\nu' = (\nu + T + \tau - T')[R \leftarrow 0]$ and $v' = v[R \leftarrow T + \tau - \epsilon]$. We have $d_\infty(\nu', T + \tau - v') \leq \Delta_t$. In fact, for any clock $x \notin R$, this follows from $d_\infty(\nu + T - T', T - v) \leq \Delta_t$, and for all $x \in R$, we have $\nu'(x) = 0$ and $(T + \tau - v')(x) = \epsilon \leq \Delta_t$. □

For any timed automaton $\mathcal{A}$ and $\Delta' \geq 0$, we define $[\![\mathcal{A}]\!]^{\geq \Delta'}$ as a TTS whose states are (ℓ, ν, u), where l is a location, ν a clock valuation and u the time elapsed since the latest action (and it is 0 initially).

Lemma 11.3.2. *Let $\mathcal{A}$ be a closed timed automaton, and $\Delta_r, \Delta_c, \Delta_t > 0$ parameters. Then, for any $\Delta \geq 2\Delta_c + 4\Delta_t$, and any scheduler ρ,*

$$[\![\mathcal{A}]\!]^{\geq 2\Delta_r + \Delta_t} \sqsubseteq [\![\mathcal{A}_\Delta]\!]_\rho^{\mathit{Impl}}.$$

Proof. We show that the relation $\mathcal{R}$ defined by $(\ell, \nu, T)\mathcal{R}(\ell, T', v, u_0)$ such that $T' \in [T, T + \Delta_t]$ and $d_\infty(\nu + (T' - T), T' - v) \leq \Delta_t$ is a timed simulation. Consider such a pair of states.

Suppose that $(\ell, \nu, T) \xrightarrow{\sigma(T+\tau)} (\ell', \nu', T + \tau)$ for some $\sigma \in \Sigma_{\mathsf{in}}$, where $\nu' = (\nu + \tau)[R \leftarrow 0]$. For any $(u_0, \epsilon) \in [0, \Delta_r] \times [0, \Delta_t]$ given by ρ, this is simulated by the following transition.

$$(\ell, T', v, u_0) \xrightarrow{\sigma(T+\tau)} (\ell', T + \tau + \epsilon, v[R \leftarrow T + \tau + \epsilon], u_0'),$$

In fact, by hypothesis, $\tau \geq 2\Delta_r + \Delta_t$, so $(T + \tau) - T' \geq \Delta_r \geq u_0$, hence the reaction time is over when the action happens. Let us show that the guard is satisfied. We have $\nu + \tau \models g$. Since $d_\infty(\nu + T' - T, T' - v) \leq \Delta_t$, we have $d_\infty(\nu + \tau, T + \tau - v) \leq \Delta_t$ (in fact, we add $\tau - (T' - T) = \tau - T' + T$ to the first vector, and $(T + \tau) - T' = \tau - T' + T$ to the second). Hence $T + \tau + \epsilon - v \models \langle g \rangle_{4\Delta_t}$. Then, $\lfloor T + \tau + \epsilon \rfloor_{\Delta_c} - \lfloor v \rfloor_{\Delta_c} \models \langle g \rangle_{4\Delta_t + 2\Delta_c}$, so the guard is satisfied. Let us write $v' = v[R \leftarrow T + \tau + \epsilon]$. It remans to show that $d_\infty(\nu' + \epsilon, T + \tau + \epsilon - v') \leq \Delta_t$. In fact, for all $x \notin R$ this follows from hypothesis since the difference between the values of a clock in two systems is unchanged when both systems delay to time instant $T + \tau + \epsilon$. For all $x \in R$, we have $(\nu' + \epsilon)(x) = \epsilon \leq \Delta_t$ and $(T + \tau + \epsilon - v')(x) = 0$. Hence, $(\ell', \nu', T + \tau)\mathcal{R}(\ell', T + \tau + \epsilon, v', u_0')$.

Suppose now that $(\ell, \nu, T) \xrightarrow{\sigma(T+\tau)} (\ell', \nu', T + \tau)$ for some $\sigma \in \Sigma_{\mathsf{out}}$, where $\nu' = (\nu + \tau)[R \leftarrow 0]$. For any $(u_0', \epsilon) \in [0, \Delta_r] \times [0, \Delta_t]$ given by ρ, this is simulated by the following.

$$(\ell, T', v, u_0) \xrightarrow{\sigma(T+\tau)} (\ell', T + \tau, v[R \leftarrow T + \tau - \epsilon], u_0'),$$

In fact, by hypothesis, $\tau \geq 2\Delta_r + \Delta_t$, so $T + \tau - \epsilon - T' \geq \Delta_r$. Let us show that the guard is satisfied. We have $\nu + \tau \models g$. Since $d_\infty(\nu + (T' - T), T' - v) \leq \Delta_t$, we have $d_\infty(\nu + \tau, T + \tau - v) \leq \Delta_t$ as in the previous case, hence $T + \tau - \epsilon - v \models \langle g \rangle_{4\Delta_t}$. Then, $\lfloor T + \tau - \epsilon \rfloor_{\Delta_c} - \lfloor v \rfloor_{\Delta_c} \models \langle g \rangle_{4\Delta_t + 2\Delta_c}$. It remans to show that $d_\infty(\nu', T + \tau - v') \leq \Delta_t$. This follows from hypothesis for clocks $x \notin R$ since both systems delay to time instant $T + \tau$. For all $x \in R$, we have $(\nu')(x) = 0$ and $(T + \tau - v')(x) = \epsilon \leq \Delta_t$. Hence, $(\ell', \nu', T + \tau)\mathcal{R}(\ell', T + \tau, v', u_0')$. □

We are now interested in the preservation of non-blockingness in the implementation semantics. Note that this is not a consequence of the simulation relations. We first prove a property on the enlarged zones.

Definition 11.3.3. *Let Z be any closed convex subset of $\mathbb{R}^{\mathcal{C}}_{\geq 0}$. The* lower boundary *of Z is the set $lb(Z) = \{v \in Z \mid \forall \tau > 0, v - \tau \notin Z\}$. The* width *of Z is defined as $\inf\{\tau \mid \exists v \in lb(Z), v + \tau \notin Z\}$. In other terms, the width of Z is the least delay necessary to go out of Z starting inside the lower boundary.*

A guard is said to be normalized if the corresponding DBM is normalized. Recall that all guards we consider are closed, thus so are DBMs.

Lemma 11.3.4. *Let g be a normalized guard such that $[\![g]\!] \neq \emptyset$. Then for any $\Delta > 0$, $[\![\langle g \rangle_\Delta]\!]$ has width greater than or equal to Δ.*

Proof. Let M be a DBM that describes $[\![g]\!]$. Then $M' = M + \Delta\mathbf{1}$ describes $[\![\langle g \rangle_\Delta]\!]$, where $\mathbf{1}$ is the matrix with same dimension as M in which all coefficients are 1's. Let $v \in lb([\![M']\!])$. We will show that $v + \tau \in [\![M']\!]$ for all $\tau \in [0, \Delta]$. First, observe that clock differences are constant during delay transitions. So, whenever $-\mathsf{M}_{y,x} - \Delta \leq v(x) - v(y) \leq \mathsf{M}_{x,y} + \Delta$, we have $-\mathsf{M}_{y,x} - \Delta \leq (v(x) + \tau) - (v(y) + \tau) \leq \mathsf{M}_{x,y} + \Delta$ for all τ. We now show that rectangular constraints

are also satisfied for at least Δ time units. Since $v \in lb([\![M']\!])$, there exists $x \in \mathcal{C}$, such that $-\mathsf{M}_{0,x} - \Delta = v(x)$ (otherwise valuation v can be decremented by some positive amount). For this clock, obviously $-\mathsf{M}_{0,x} - \Delta \leq v(x) + \tau \leq \mathsf{M}_{x,0} + \Delta$ for $\tau \in [0, \Delta]$, since $-\mathsf{M}_{0,x} \leq \mathsf{M}_{x,0}$ (in fact, the set is not empty). Now, consider any $y \in \mathcal{C}$. DBM M' implies the following diagonal constraint (and possibly a tighter one).

$$-\Delta - \mathsf{M}_{0,x} - \mathsf{M}_{y,0} \leq v(x) - v(y) \leq \mathsf{M}_{x,0} + \mathsf{M}_{0,y} + \Delta.$$

But, combining the above inequality with $v(x) = -\mathsf{M}_{0,x} - \Delta$, we get that $v(y) \leq \mathsf{M}_{y,0}$, so $v(y) + \tau \leq \mathsf{M}'_{y,0} = \mathsf{M}_{y,0} + \Delta$ for $\tau \in [0, \Delta]$. □

The following lemma shows that if the exact semantics is non-blocking, then the implementation semantics is also non-blocking.

Lemma 11.3.5. *Let $\mathcal{A}$ be a closed timed automaton, $\Delta_r, \Delta_c, \Delta_t > 0$ be parameters and assume that $[\![\mathcal{A}]\!]$ is non-blocking. Then for any $\Delta \geq \Delta_r + 2\Delta_c + \Delta_t$, $[\![\mathcal{A}_\Delta]\!]^{\mathsf{Impl}}$ is non-blocking.*

Proof. First, observe that if $[\![\mathcal{A}]\!]$ is non-blocking, then so is $[\![\mathcal{A}_\Delta]\!]$ for any $\Delta > 0$. In fact, consider an edge $\ell \xrightarrow{g,\sigma,R} \ell'$ and any corresponding transition $(\ell, v_1, T) \xrightarrow{\sigma(T+\tau)} (\ell', v_1', T+\tau)$ in $[\![\mathcal{A}_\Delta]\!]$. Since $v_1 + \tau \models_\Delta g$, there exists v_2 such that $d_\infty(v_2, v_1) \leq \Delta$ and $v_2 + \tau \models g$. Let v_2' be such that $(\ell, v_2, T) \xrightarrow{\sigma(T+\tau)} (\ell', v_2', T+\tau)$. Since $[\![\mathcal{A}]\!]$ is non-blocking, for some $\tau \geq 0$, there exists an edge with guard g' such that $v_2' + \tau \models g'$. But $d_\infty(v_1', v_2') \leq \Delta$, so we also have $v_1' + \tau \models_\Delta g'$, and the edge is enabled from $(\ell', v_1', T+\tau)$ as well. Hence $[\![\mathcal{A}_\Delta]\!]$ is non-blocking.

Let $(\ell, T, v, u_0) \xrightarrow{\sigma(T+\tau)} (\ell', T+\tau+\epsilon, v', u_0')$ denote a transition in $[\![\mathcal{A}_\Delta]\!]^{\mathsf{Impl}}$ with $v' = v[R \leftarrow T+\tau+\epsilon]$ for some $\sigma \in \Sigma_{\mathsf{in}}$, and $\ell \xrightarrow{\sigma,g,R} \ell'$ the corresponding edge in $\mathcal{A}$. We have $\lfloor T+\tau+\epsilon \rfloor_{\Delta_c} - \lfloor v \rfloor_{\Delta_c} \models_\Delta g$, so for $\nu = \lfloor T+\tau+\epsilon \rfloor_{\Delta_c} - \lfloor v \rfloor_{\Delta_c}$, we have $(\ell, \nu, 0) \xrightarrow{\sigma(0)} (\ell', \nu', 0)$ in $[\![\mathcal{A}_\Delta]\!]$, where $\nu' = \nu[R \leftarrow 0]$. Notice that $\nu' = \lfloor T+\tau+\epsilon \rfloor_{\Delta_c} - \lfloor v' \rfloor_{\Delta_c}$. Since $[\![\mathcal{A}_\Delta]\!]$ is non-blocking, there exists $\tau \geq 0$ such that $\nu' + \tau \models_\Delta g'$, where g' is the guard of some edge with label σ' outgoing from ℓ'.

Consider $Z = [\![\langle g' \rangle_\Delta]\!]$, whose width is at least Δ by Lemma 11.3.4. Then, there exists $\alpha, \beta \in \mathbb{R}_{\geq 0}$ such that $\alpha + \Delta \leq \beta$ and $\nu' + \tau' \in [\![\langle g' \rangle_\Delta]\!]$ for all $\tau' \in [\alpha, \beta]$. We show that $(\ell', T+\tau+\epsilon, v', u_0')$ can delay some amount τ' and take this transition. For any $\max(\alpha, \Delta_r) + \Delta_c \leq \tau' \leq \beta - \Delta_c - \Delta_t$, we have

$$\begin{aligned} \lfloor T+\tau+\epsilon \rfloor_{\Delta_c} + (\max(\alpha, \Delta_r) + \Delta_c) - \Delta_c &\leq \lfloor T+\tau+\epsilon+\tau' \rfloor_{\Delta_c} \\ &\leq \lfloor T+\tau+\epsilon \rfloor_{\Delta_c} + \beta - \Delta_c - \Delta_t + \Delta_c. \end{aligned}$$

Thus,

$$\nu' + \max(\alpha, \Delta_r) \leq \lfloor T+\tau+\epsilon+\tau' \rfloor_{\Delta_c} - \lfloor v' \rfloor_{\Delta_c} \leq \nu' + \beta - \Delta_t. \tag{11.1}$$

Notice that such a τ' exists since $\Delta_r + 2\Delta_c + \Delta_t \leq \Delta$. Hence, if $\sigma' \in \Sigma_{\mathsf{out}}$, then, for any $\epsilon' \in [0, \Delta_t]$ given by ρ, the transition

$$(\ell', T+\tau+\epsilon, v', u_0') \xrightarrow{\sigma'(T+\tau+\epsilon+\tau'+\epsilon')} (\ell'', T+\tau+\epsilon+\tau'+\epsilon', \cdot, \cdot),$$

is valid. If $\sigma' \in \Sigma_{\mathsf{in}}$, then

$$(\ell', T+\tau+\epsilon, v', u_0') \xrightarrow{\sigma'(T+\tau+\epsilon+\tau')} (\ell'', T+\tau+\epsilon+\tau'+\epsilon', \cdot, \cdot)$$

is valid thanks to the right hand side term in (11.1) (since $\epsilon' \in [0, \Delta_t]$.

The proof is similar when $\sigma \in \Sigma_{\mathsf{out}}$. □

11.4 Comparison with [DDR05a]

A similar semantics, called the *program semantics*, was defined in [DDR05a] and was proven to be simulated by the enlarged semantics (as in the rightmost simulation in Proposition 11.2.2). Our definition follows their ideas, but we define a somewhat more abstract semantics by concentrating on different aspects. Our goal here in this chapter is not defining the semantics of a microprocessor model but rather introducing various kinds of imprecisions. One difference with [DDR05a] is that we do not insist on the semantics to produce events *almost as soon as possible (almost ASAP)*. Thus, transitions are not urgent in our semantics; only the reactions to input events are. Our semantics is not *input-enabled*, that is, it can ignore signals during the treatment of another signal, since it has no buffer. Both assumptions are applicable to different platforms (see [AMP98, Die01] for examples of systems that ignore any signal unless it is maintained long enough). Moreover, instead of giving a detailed model of the treatment of the signals in several steps, we rather define action transitions taking a positive unpredictable amount of time, during which computations take place. This allows us to model the unpredictability using schedulers and state our properties *for any scheduler*. Due to these differences, both semantics are uncomparable: the program semantics is not simulated, in general, by our implementation semantics, and vice versa. Note that two results in Proposition 11.2.2 are new compared to [DDR05a]: the leftmost simulation and the preservation of non-blockingness.

Part V

Conclusion

Chapter 12

Conclusion and Perspectives

We summarize here the results presented in this thesis. Conclusions and possible future work were given specifically at the end of each chapter. We mention here more general perspectives for future research directions.

Robustness analysis One of the important perturbation models considered in the literature is that of guard enlargement. We studied the untimed languages of timed automata under parameterized guard enlargement. We gave an algorithm to decide whether the untimed language of a given timed automaton is preserved under some enlargement parameter. This extends the possibility of robustness analysis: while previous work allowed to do model-checking for a given property, under a parameterized enlargement parameter, our algorithm allows to check for complete behavior preservation. The algorithm runs in exponential space, which matches the complexity of checking untimed language equivalence between timed automata.

The semantics of timed automata under parameterized guard enlargement have thus been studied for several properties. Although this is a parameter synthesis problem, one can notice that these robustness problems have no complexity cost over the corresponding standard verification problems. We believe efforts in this field should now concentrate on developing efficient algorithms, so that robust model-checking becomes a standard feature in timed automata model-checkers. Some symbolic algorithms have been developed for this aim, but they only apply to a restricted class of timed automata, namely timed automata without nested loops [DK06, JR11]. Extending these techniques to general timed automata remains an open problem. A timed automata designer interested in robustness analysis should note that in case the parameterized robustness problems are too inefficient, it is always possible to syntactically enlarge the guards for a fixed (guessed) enlargement parameter, and model-check the resulting timed automaton. Although there is no guarantee on termination, one could stop trying when enlargement parameter becomes too small. This would work on models that are not too large, and has the advantage of being simple. But for large models with large constants, this procedure could easily become impractical. An important question here is whether, and to what extent parameterized robustness analysis algorithms (against guard enlargements) overperform a simple binary search on enlargement parameters.

An interesting future direction is understanding whether the similarity between the semantics under clock drifts and guard enlargements can be extended beyond safety (these were shown to be equivalent for safety in [Pur00, DDMR08]). For instance, could parameterized robust model-checking algorithms for ω-regular objectives, or untimed language preservation be adapted to clock drifts? Does correctness under guard enlargement imply correctness under clock drifts? Such results would have practical significance since one can model-check under a fixed guard enlargement with existing tools for timed automata whereas clock drifts require hybrid automata model-checkers.

In this thesis, we also introduced a new notion of robustness, namely shrinkability, based on syntactic perturbations of the guards. Shrinking is the inverse operation of the enlargement, that is,

tightening the guards so as to disallow behaviors that are in the limit of satisfying the guards. We were interested in synthesizing possibly different shrinking parameters for all guards, under which the resulting timed automaton is non-blocking and time-abstract similar to a given finite automaton. Thus, while analysis under guard enlargements was concerned with the newly appeared behaviors, here the emphasis is on the preservation of time-abstract behavior under removal of limit (timed) behavior. The resulting problems are decidable, with reasonable theoretical complexity (these vary from NP to EXPTIME). Shrinkability has also implications in implementability of timed automata models. In fact, to overcome th effect of imprecise time measures (that is, guard enlargements), one can shrink a timed automaton, then check whether any important time-abstract behavior is lost. We presented a software tool in which a shrinkability algorithm is implemented. We were able to treat several case studies from the literature, where timed automata models had, *e.g.* upto 11 clocks and hundreds of transitions (the train gate model), or 4 clocks with thousands of transitions (Fischer's protocol).

We plan to extend the theory of fixpoint equations between shrunk DBMs by studying the arbitrary use of the union. This would allow expressing much richer properties, such as reachability, simulation (without the hypothesis of distinct edges), and one could even treat temporal logic formulae. This would give the system designer much freedom when applying shrinkability analysis, compared to the actual algorithms which require specifying a finite automaton with respect to which simulation is checked. In order to make shrinkability analysis appealing, further tool support is also needed to interpret and refine the finite automaton specification.

Note that for shrinkability analysis, manually trying different shrinking parameters is not an option, in contrast with the robustness analysis against enlargements. In fact, the shrinking parameters have dependencies between them (which we expressed as max-plus graphs) and these are difficult to solve by hand even for small models. Thus, automatic synthesis of these parameters are an essential feature in shrinkability algorithms.

Robust synthesis Robustness analysis algorithms are used to check whether a timed automaton design preserves its behavior, or satisfies its specification, under perturbations. Thus, when applying these algorithms, one assumes that the given timed automaton is designed to do exactly what one expects it to do. Another approach in formal verification is that of controller synthesis. Here, the given model is often *under-specified*, that is, it contains too much non-determinism, or several ways of performing similar behaviors. Given such a model, and a specification, the goal is then to find a strategy to guide the system so as to satisfy the specification. The model controlled by the strategy is often deterministic, so it can be seen as an implementation.

In this thesis, we investigated the controller synthesis problems on timed automata and games under robustness concerns. We modelled the perturbations as a game between a controller and an environment (these were called Controller, and Perturbator). In this semantics, both players suggest valid delays and actions. Moreover, if Controller's move is taken, Perturbator perturbs the delay by a parameterized amount. We considered two variants. In the excess perturbation game, Controller only needs to suggest moves that satisfy the guards; while in the conservative perturbation game, the moves should satisfy the guards under any perturbation. The conservative perturbation game, where the magnitude of the perturbations is seen as a parameter, yields to a PSPACE-complete problem for Büchi objectives in timed automata. The proofs establish a strong relation between controllability in our sense and the absence of convergence phenomena. The excess perturbation game is more complex: reachability is EXPTIME-complete already on timed automata; we also extend this algorithm to turn-based timed games. We apply these game semantics to weighted timed

games, on which the optimal-cost reachability problem is known to be undecidable. We proved that, unfortunately, the problem remains undecidable under both semantics.

We believe that both semantics are interesting under different modelling assumptions. We have seen that while the excess perturbation game allow to keep the model simple, but introduce additional behaviors, the conservative perturbation game semantics requires the system designer to define strict constraints on the behaviors, and then concentrate on satisfying these. In both semantics, we formulate the control problems from a worst-case analysis point of view. In fact, the semantics are based on the assumption that the perturbations are controlled by an adversary with an opposite objective. A robust system, in the sense of these semantics, is then capable of supporting any source of perturbations.

An important direction is the treatment of general timed games. Although both semantics seem plausible for modelling purposes, we will privilege the conservative game semantics, since it yields problems with lower complexity. Technically, both algorithms use shrunk DBMs, which we also used to solve shrinkability problems. We believe this is a good data structure to explore the parameterized state space of timed automata, and we expect it to be used in other problems related to robustness in timed systems.

Once the theory of parameterized perturbation game semantics has been fully understood, the following research goal will be finding symbolic algorithms efficient enough to allow treating real case studies. One could hope to extend the on-the-fly algorithm of [CDF$^+$05] to one of our perturbation game semantics. The long term goal should be to integrate robust controller synthesis algorithms in timed game solvers, such as Uppaal-TIGA [BDL$^+$06] and Synthia [PEM11]. As in the case of robustness analysis, the relevance of computing the perturbation parameter should be evaluated compared to manually testing different values in terms of performance.

We studied perturbation game semantics in weighted timed automata and games, and gave a PSPACE algorithm for cost-optimal reachability in weighted timed automata under the conservative perturbation game semantics. On the other hand, these semantics unfortunately do not render cost-optimal reachability problems decidable on weighted timed games. In order to obtain decidability, one could restrict to closed weighted timed games. In fact, closed guards make it difficult for players to check whether the complement of an expected behavior has been performed. Similarly, another idea is to look at different semantics which eliminate "isolated" behaviors, such as probabilistic or topological semantics.

Robust implementation A third set of results we considered here is on implementation, that is, automatically transforming a given timed automaton so that it can be implemented. We gave constructions to transform any given timed automaton into an equivalent one, in the sense of strong bisimulation, whose behavior under enlargement is approximately bisimilar to its exact behavior. It follows that the enlargement of the transformed timed automaton is approximately bisimilar to the original timed automaton. We also gave a variant which only preserves safety properties, but may not be time-abstract bisimilar with the original timed automaton. These results enable a design methodology that clearly separates design and problems related to robustness. In fact, one can design a system in the exact semantics, apply model-checking, and then automatically "compile" to an equivalent and robust model. Hence, in this development procedure, there is no need for the robustness analysis, since it is ensured by construction.

We also considered a more concrete target implementation semantics for the implementation problem. We described a semantics with a discrete clock, positive communication and reaction times, and proved that the behaviors of shrinkable timed automata are preserved under this semantics. In

this approach, the resulting semantics is (exactly) simulated by the original timed automaton, in contrast with the approximate bisimulation with the former results. So this method ensures strict behavior preservation. However not all timed automata are shrinkable, so one cannot implement all timed automata in this framework.

In future, we plan to extend these "robust-by-construction" design techniques. For instance, similar results need to be adapted to networks of timed automata, where the transformed components do not need to access other components' clocks. For these to be applicable in general, the theory also needs to be extended to timed games, so that one distinguishes between controllable and uncontrollable transitions.

General Remarks In this thesis, we considered several perturbation models, each reflecting a different point of view on the nature of the perturbations. One could wonder which one is the "best" notion of robustness, and deserves more attention for future research. Similar questions could be asked by a timed automaton practitioner who wants to check the robustness of their timed automaton design.

We believe that the theory of robustness in timed automata naturally needs this diversity in perturbation models. In fact, timed automata are a general-purpose formalism: Its applications range from low-level circuits to high-level schedulings of multimedia tasks, intended to be implemented as a monolithic system or a distributed one. Thus, the system designer needs to determine which kind of robustness analysis makes sense given the application, say, among time measurement errors (guard enlargement), clock synchronization errors (clock drifts), or sampling. A rich theory of robustness will allow the system designer to pick the solution that is the most adapted to the application at hand.

A common practice in engineering is decomposing a complex task into several simpler pieces that are easier to solve, then obtaining a global solution by combining the small solutions. This approach can also be found in formal verification, where smaller components are verified under some assumptions, then recombined to prove the correctness of the global system. Such a decomposition allows the treatment of large systems.

Concerning robustness, an important step would be to find decidable criteria on timed automata and their synchronization which ensure that their combination (network of timed automata) is robust by construction. This does not seem easy, since even very simple and robust timed automata may define non-robust timed automata once combined by synchronization (This is the case of Fig. 2.1). Thus, non-trivial criteria might be needed, including the restriction of synchronization between components in order to define a "robust composition operator". Possible applications would be analyzing the robustness of each component separately, and deducing the robustness of the global system, or applying a robust implementation construction on each component such that their composition is robust by construction.

Bibliography

[AAM06] Yasmina Adbeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006.

[AB01] Eugene Asarin and Ahmed Bouajjani. Perturbed turing machines and hybrid systems. In *Logic in Computer Science, 2001. Proceedings. 16th Annual IEEE Symposium on*, pages 269–278. IEEE, 2001.

[ABG+08] S. Akshay, Benedikt Bollig, Paul Gastin, Madhavan Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. In Franck van Breugel and Marsha Chechik, editors, *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, volume 5201 of *Lecture Notes in Computer Science*, pages 82–97, Toronto, Canada, August 2008. Springer.

[ABM04] Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 122–133. Springer Berlin Heidelberg, 2004.

[ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, May 1993.

[ACS10] Tesnim Abdellatif, Jacques Combaz, and Joseph Sifakis. Model-based implementation of real-time applications. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 229–238, New York, NY, USA, 2010. ACM.

[AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Michael S. Paterson, editor, *Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer Berlin Heidelberg, 1990.

[AD92] Rajeev Alur and David L. Dill. The theory of timed automata. In J.W. Bakker, C. Huizing, W.P. Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 45–73. Springer Berlin Heidelberg, 1992.

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

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211(1-2):253 – 273, 1999.

[AFKS12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In Dimitra Giannakopoulou and Dominique Méry, editors, *Proceedings of the 18th International Symposium on Formal Methods (FM'12)*, volume 7436 of *Lecture Notes in Computer Science*. Springer, 2012.

[AFM+02] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times - a tool for modelling and implementation of embedded systems. In *TACAS*, pages 460–464, 2002.

[AHJR12] S. Akshay, Loïc Hélouët, Claude Jard, and Pierre-Alain Reynier. Robustness of time petri nets under guard enlargement. In Alain Finkel, Jérôme Leroux, and Igor Potapov, editors, *Reachability Problems*, volume 7550 of *Lecture Notes in Computer Science*, pages 92–106. Springer Berlin Heidelberg, 2012.

[AIK+03] Rajeev Alur, Franjo Ivancic, Jesung Kim, Insup Lee, and Oleg Sokolsky. Generating embedded software from hierarchical hybrid models. *SIGPLAN Not.*, 38(7):171–182, June 2003.

[AKY07] Parosh Abdulla, Pavel Krčál, and Wang Yi. Sampled universality of timed automata. In *Proc. 10th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'07)*, volume 4423 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2007.

[AKY10] Parosh Abdulla, Pavel Krčál, and Wang Yi. Sampled semantics of timed automata. *Logical Methods in Computer Science*, 6(3:14), 2010.

[ALM05] Rajeev Alur, Salvatore LaTorre, and P. Madhusudan. Perturbed timed automata. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, volume 3414 of *Lecture Notes in Computer Science*, pages 70–85. Springer, 2005.

[AMP95] Eugene Asarin, Oded Maler, and Amir Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II*, volume 999 of *LNCS*, pages 1–20. Springer, 1995.

[AMP98] Eugene Asarin, Oded Maler, and Amir Pnueli. On discretization of delays in timed automata and digital circuits. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR'98 Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 470–484. Springer, 1998.

[AMPS98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proceedings of the 5th IFAC Cconference on System Structure and Control (SSC'98)*, pages 469–474. Elsevier Science, 1998.

[AT05] Karine Altisen and Stavros Tripakis. Implementation of timed automata: An issue of semantics or modeling? In Paul Pettersson and Wang Yi, editors, *Proceedings of the 3rd International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2005.

[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, pages 49–62, London, UK, UK, 2001. Springer-Verlag.

[BA11] Nicolas Basset and Eugene Asarin. Thin and thick timed regular languages. In Uli Fahrenberg and Stavros Tripakis, editors, *Formal Modeling and Analysis of Timed Systems*, volume 6919 of *Lecture Notes in Computer Science*, pages 113–128. Springer Berlin Heidelberg, 2011.

[BBBR07] Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem on weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, October 2007.

[BBM06] Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5):188–194, June 2006.

[BBR05a] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *Proceedings of the Third international conference on Formal Modeling and Analysis of Timed Systems*, FORMATS'05, pages 49–64, Berlin, Heidelberg, 2005. Springer-Verlag.

[BBR05b] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *Proceedings of the Third international conference on Formal Modeling and Analysis of Timed Systems*, FORMATS'05, pages 49–64, Berlin, Heidelberg, 2005. Springer-Verlag.

[BC05] Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 10(4):393–405, 2005.

[BCFL04] Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal strategies in priced timed game automata. In Kamal Lodaya and Meena Mahajan, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer Berlin Heidelberg, 2004.

[BCOQ92] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and Linearity – An Algebra For Discrete Event Systems.* John Wiley & Sons, 1992.

[BDFP00] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are timed automata updatable? In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479, Chicago, Illinois, USA, July 2000. Springer.

[BDL+06] Gerd Behrmann, Alexandre David, Kim .G. Larsen, John Hakansson, Paul Petterson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *Quantitative Evaluation of Systems, 2006. QEST 2006. Third International Conference on*, pages 125–126. IEEE, 2006.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 546–550, London, UK, 1998. Springer-Verlag.

[Ber00] Gérard Berry. The foundations of Esterel. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction – Essays in Honour of Robin Milner*, pages 425–454. MIT Press, 2000.

[BFH+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced time automata. In MariaDomenica Benedetto and Alberto Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer Berlin Heidelberg, 2001.

[BGHS81] J. Bermond, A. Germa, M. Heydemann, and D. Sotteau. Longest paths in digraphs. *Combinatorica*, 1:337–341, 1981.

[BGS12] Romain Brenguier, Stefan Göller, and Ocan Sankur. A comparison of succinctly represented finite-state systems. In Maciej Koutny and Irek Ulidowski, editors, *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR'12)*, volume 7454 of *Lecture Notes in Computer Science*, pages 147–161, Newcastle, UK, September 2012. Springer.

[BKW12] Victor Bandur, Wolfram Kahl, and Alan Wassyng. Microcontroller assembly synthesis from timed automaton task specifications. In Mariëlle Stoelinga and Ralf Pinger, editors, *Formal Methods for Industrial Critical Systems*, volume 7437 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2012.

[BLM+11] Patricia Bouyer, Kim G. Larsen, Nicolas Markey, Ocan Sankur, and Claus Thrane. Timed automata can always be made implementable. In Joost-Pieter Katoen and Barbara König, editors, *Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR'11)*, volume 6901 of *Lecture Notes in Computer Science*, pages 76–91, Aachen, Germany, September 2011. Springer.

[BLN03] Dirk Beyer, Claus Lewerentz, and Andreas Noack. Rabbit: A tool for bdd-based verification of real-time systems. In Jr. Hunt, WarrenA. and Fabio Somenzi, editors, *Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 122–125. Springer Berlin Heidelberg, 2003.

[BM83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In R. E. A. Mason, editor, *Information Processing 83 – Proceedings of the 9th IFIP World Computer Congress (WCC'83)*, pages 41–46. North-Holland/IFIP, September 1983.

[BMOW07] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. The cost of punctuality. In *Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science (LICS'07)*, pages 109–118, Wrocław, Poland, July 2007. IEEE Computer Society Press.

[BMPY97] Marius Bozga, Oded Maler, Amir Pnueli, and Sergio Yovine. Some progress in the symbolic verification of timed automata. In *Proc. 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 1997.

[BMR06] Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust model-checking of linear-time properties in timed automata. In José R. Correa, Alejandro Hevia, and Marcos Kiwi, editors, *Proceedings of the 7th Latin American Symposium on Theoretical INformatics (LATIN'06)*, volume 3887 of *Lecture Notes in Computer Science*, pages 238–249. Springer, 2006.

[BMR08] Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust analysis of timed automata via channel machines. In Roberto Amadio, editor, *Proceedings of the 11th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'08)*, volume 4962 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2008.

[BMS11] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust model-checking of timed automata via pumping in channel machines. In Uli Fahrenberg and Stavros Tripakis, editors, *Proceedings of the 9th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'11)*, volume 6919 of *Lecture Notes in Computer Science*, pages 97–112, Aalborg, Denmark, September 2011. Springer.

[BMS12] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust reachability in timed automata: A game-based approach. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP'12) – Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 128–140. Springer, July 2012.

[BY04] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 2098 of *Lecture Notes in Computer Science*, pages 87–124. Springer-Verlag, 2004.

[CDF+05] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Lmei Didier. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 2005*, pages 66–80, London, UK, 2005. Springer-Verlag.

[CHP11] Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Timed parity games: Complexity and robustness. *Logical Methods in Computer Science*, 7(4), 2011.

[CHR02] Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In Claire Tomlin and Mark R. Greenstreet, editors, *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.

[CM05] Paul Caspi and Oded Maler. From control loops to real-time programs. *Handbook of networked and embedded control systems*, pages 395–418, 2005.

[DDMR08] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robust safety of timed automata. *Formal Methods in System Design*, 33(1-3):45–84, 2008.

[DDR05a] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics: From timed models to timed implementations. *Formal Aspects of Computing*, 17(3):319–341, 2005.

[DDR05b] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Systematic implementation of real-time models. In John Fitzgerald, IanJ. Hayes, and Andrzej Tarlecki, editors, *FM 2005: Formal Methods*, volume 3582 of *Lecture Notes in Computer Science*, pages 139–156. Springer Berlin Heidelberg, 2005.

[DHLP06] Alexandre David, John Håkansson, Kim Gulstrand Larsen, and Paul Pettersson. Model checking timed automata with priorities using DBM subtraction. In Eugene Asarin and Patricia Bouyer, editors, *Proceedings of the 4th International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2006.

[Die01] Henning Dierks. PLC-automata: a new class of implementable real-time automata. *Theoretical Computer Science*, 253:61–93, 2001.

[Dil90] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In Joseph Sifakis, editor, *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems (AVMFSS'89)*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1990.

[Dim07] Cătălin Dima. Dynamical properties of timed automata revisited. In Jean-François Raskin and P. Thiagarajan, editors, *Formal Modeling and Analysis of Timed Systems*, volume 4763 of *Lecture Notes in Computer Science*, pages 130–146. Springer Berlin / Heidelberg, 2007.

[DK06] Conrado Daws and Piotr Kordy. Symbolic robustness analysis of timed automata. In Eugene Asarin and Patricia Bouyer, editors, *Proceedings of the 4th International Conferences on Formal Modelling and Analysis of Timed Systems, (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 143–155. Springer, 2006.

[DL07] Cătălin Dima and Ruggero Lanotte. Distributed time-asynchronous automata. In *Proceedings of the 4th international conference on Theoretical aspects of computing*, ICTAC'07, pages 185–200, Berlin, Heidelberg, 2007. Springer-Verlag.

[DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with kronos. In *In Proc. 1995 IEEE Real-Time Systems Symposium, RTSS'95*, pages 66–75. IEEE Computer Society Press, 1995.

[DY00] Alexandre David and Wang Yi. Modelling and analysis of a commercial field bus protocol. In *Proceedings of the 12th Euromicro Conference on Real Time Systems*, pages 165–172. IEEE Computer Society, 2000.

[FLT10] Uli Fahrenberg, Kim G. Larsen, and Claus Thrane. A quantitative characterization of weighted Kripke structures in temporal logic. *Journal of Computing and Informatics*, 29(6+):1311–1324, 2010.

[Frä99] Martin Fränzle. Analysis of hybrid systems: An ounce of realism can save an infinity of states. In Jörg Flum and Mario Rodriguez-Artalejo, editors, *Computer Science Logic*, volume 1683 of *Lecture Notes in Computer Science*, pages 126–139. Springer, 1999.

[GHJ97] Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan. Robust timed automata. In *Proc. International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 331–345. Springer, 1997.

[GV08] Orna Grumberg and Helmut Veith, editors. *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*. Springer, 2008.

[Hen08] Thomas A. Henzinger. Two challenges in embedded systems design: Predictability and robustness. *Philosophical Transactions of the Royal Society*, 366:3727–3736, 2008.

[HHK01] Thomas A. Henzinger, Benjamin Horowitz, and Christoph M. Kirsch. Giotto: A time-triggered language for embedded programming. In *EMSOFT*, volume 2211 of *Lecture Notes in Computer Science*, pages 166–184. Springer, 2001.

[HKSW11] Frédéric Herbreteau, Dileep Kini, B. Srivathsan, and Igor Walukiewicz. Using non-convex approximations for efficient analysis of timed automata: Extended version. Technical Report cs.LO/1110.3704v1, arXiv – Computing Research Repository, 2011.

[HLY92] Uno Holmer, Kim Larsen, and Wang Yi. Deciding properties of regular real timed processes. In Kim G. Larsen and Arne Skou, editors, *Computer Aided Verification*, volume 575 of *Lecture Notes in Computer Science*, pages 443–453. Springer Berlin Heidelberg, 1992.

[HMP92] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, ICALP '92, pages 545–558, London, UK, UK, 1992. Springer-Verlag.

[HR00] Thomas A. Henzinger and Jean-François Raskin. Robust undecidability of timed and hybrid systems. In Nancy Lynch and BruceH. Krogh, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 145–159. Springer Berlin Heidelberg, 2000.

[HRSV01] Thomas Hune, Judi Romijn, Mariëlle Stœlinga, and Frits Vaandrager. Linear parametric model checking of timed automata. In Tiziana Margaria and Wang Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 2001.

[HS06] Thomas A. Henzinger and Joseph Sifakis. The embedded systems design challenge. In *Formal Methods, 14th International Symposium on Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15, Hamilton, Canada, 2006. Springer.

[HSLL97] Klaus Havelund, Arne Skou, Kim G. Larsen, and Kyle Lund. Formal modeling and analysis of an audio/video protocol: an industrial case study using uppaal. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, RTSS '97, pages 2–, Washington, DC, USA, 1997. IEEE Computer Society.

[JR11] Rémi Jaubert and Pierre-Alain Reynier. Quantitative robustness analysis of flat timed automata. In *Proceedings of the 14th international conference on Foundations of software science and computational structures: part of the joint European conferences on theory*

and practice of software, volume 6604 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2011.

[KLP09] Piotr Kordy, Rom Langerak, and Jan Willem Polderman. Re-verification of a lip synchronization protocol using robust reachability. In *FMA*, pages 49–62, 2009.

[Kop11] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, 2011.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2:255–299, 1990.

[KP05] Pavel Krčál and Radek Pelánek. On sampled semantics of timed systems. In *Proc. 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 310–321. Springer, 2005.

[Kri00] P. Krishnan. Distributed timed automata. *Electronic Notes in Theoretical Computer Science*, 28:5–21, 2000.

[LLTW11] Kim G. Larsen, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. Robust specification of real time components. In Uli Fahrenberg and Stavros Tripakis, editors, *Formal Modeling and Analysis of Timed Systems*, volume 6919 of *Lecture Notes in Computer Science*, pages 129–144. Springer Berlin Heidelberg, 2011.

[LS89] Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. In *Proc. 16th Annual ACM Symposium on Principles of Programming Languages*, pages 344–352, 1989.

[LY94] Kim G. Larsen and Wang Yi. Time abstracted bisimulation: Implicit specifications and decidability. In Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt, editors, *Mathematical Foundations of Programming Semantics*, volume 802 of *Lecture Notes in Computer Science*, pages 160–176. Springer Berlin Heidelberg, 1994.

[Min67] Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[OLS11] James Ortiz, Axel Legay, and Pierre-Yves Schobbens. Distributed event clock automata. In Béatrice Bouchou-Markhoff, Pascal Caron, Jean-Marc Champarnaud, and Denis Maurel, editors, *Implementation and Application of Automata*, volume 6807 of *Lecture Notes in Computer Science*, pages 250–263. Springer Berlin Heidelberg, 2011.

[OW03a] Joël Ouaknine and James Worrell. Revisiting digitization, robustness, and decidability for timed automata. In *Proc. 18th Annual Symposium on Logic in Computer Science (LICS'03)*, pages 198–207. IEEE Computer Society, 2003.

[OW03b] Joël Ouaknine and James Worrell. Universality and language inclusion for open and closed timed automata. In *Proceedings of the 6th international conference on Hybrid systems: computation and control*, HSCC'03, pages 375–388, Berlin, Heidelberg, 2003. Springer-Verlag.

[PEM11] Hans-Jörg Peter, Rüdiger Ehlers, and Robert Mattmüller. Synthia: Verification and synthesis for timed automata. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 649–655. Springer Berlin Heidelberg, 2011.

[Pur00] Anuj Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.

[Ram30] F. P. Ramsey. On a problem in formal logic. *Proc. London Math. Soc. (3)*, 30:264–286, 1930.

[RNPH05] G. Rodriguez-Navas, J. Proenza, and H. Hansson. Using uppaal to model and verify a clock synchronization protocol for the controller area network. In *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, volume 2. IEEE, 2005.

[RWT+06] Jan Reineke, BjÃűrn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. A definition and classification of timing anomalies. In *Proceedings of 6th International Workshop on Worst-Case Execution Time (WCET) Analysis*, 2006.

[San10] Ocan Sankur. Model-checking robuste des automates temporisés via les machines à canaux. Master's thesis, Ecole Normale Supérieure, 2010.

[San11] Ocan Sankur. Untimed language preservation in timed systems. In Filip Murlak and Piotr Sankowski, editors, *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS'11)*, volume 6907 of *Lecture Notes in Computer Science*, pages 556–567. Springer, August 2011.

[San13] Ocan Sankur. `Shrinktech`: A tool for the robustness analysis of timed automata. In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV'13)*. To appear, 2013.

[SBM11] Ocan Sankur, Patricia Bouyer, and Nicolas Markey. Shrinking timed automata. In Supratik Chakraborty and Amit Kumar, editors, *Proceedings of the 31st Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'11)*, volume 13 of *LIPIcs*, pages 375–386. Leibniz-Zentrum für Informatik, December 2011.

[SBMR13] Ocan Sankur, Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust controller synthesis in timed automata. In *In submission*, 2013.

[SFK08] Mani Swaminathan, Martin Fränzle, and Joost-Pieter Katoen. The surprising robustness of (closed) timed automata against clock-drift. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and Luke Ong, editors, *Fifth IFIP International Conference On Theoretical Computer Science TCS 2008*, volume 273 of *IFIP International Federation for Information Processing*, pages 537–553. Springer US, 2008.

[Sta12] Amélie Stainer. Frequencies in forgetful timed automata. In Marcin Jurdzinski and Dejan Nickovic, editors, *Formal Modeling and Analysis of Timed Systems*, volume 7595 of *Lecture Notes in Computer Science*, pages 236–251. Springer Berlin Heidelberg, 2012.

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

[TFL10] Claus Thrane, Uli Fahrenberg, and Kim G. Larsen. Quantitative analysis of weighted transition systems. *Journal Logic and Algebraic Programming*, 79(7):689–703, 2010.

[TY01] Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *Form. Methods Syst. Des.*, 18(1):25–68, January 2001.

[Wan06] Farn Wang. Redlib for the formal verification of embedded systems. In *Proceedings of the Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, ISOLA '06, pages 341–346, Washington, DC, USA, 2006. IEEE Computer Society.

Index