UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'informatique

Algorithmic Analysis of Complex Semantics for Timed and Hybrid Automata

Laurent DOYEN

Juin 2006

UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'informatique

Algorithmic Analysis of Complex Semantics for Timed and Hybrid Automata

Laurent DOYEN

Thèse présentée en vue de l'obtention du grade de Docteur en Sciences
soutenue le 13 juin 2006 devant le jury composé de:

M. Gianluca Bontempi (ULB)
Mme. Véronique Bruyère (UMH Mons, Belgique)
M. Raymond Devillers (ULB), secrétaire
M. Thomas A. Henzinger (EPFL Lausanne, Suisse et UC Berkeley, USA)
M. François Laroussinie (CNRS et ENS Cachan, France)
M. Thierry Massart (ULB), président
M. Jean-François Raskin (ULB), promoteur

Résumé

Ces quinze dernières années, beaucoup de progrès ont été réalisés dans le domaine de la vérification formelle des systèmes temps-réels. On parle de logiques, d'automates, d'algèbres de processus, de langages synchrones, etc. Dès le début, un formalisme a joué un rôle central: les *automates temporisés*, et leur extension naturelle les *automates hybrides*. Ces modèles permettent de définir des contraintes temps-réel en utilisant des *horloges* ou plus généralement des *variables continues* dont l'évolution est décrite par des équations différentielles. Ils généralisent les automates finis du fait que leur sémantique définit des *mots temporisés*, où un instant de survenance est attaché à chaque symbole discret.

La *décidabilité* et l'*analyse algorithmique* des automates temporisés et hybrides ont été abondamment étudiées dans la littérature. Le résultat principal pour les automates temporisés est qu'ils sont positivement décidables. Ce n'est pas le cas des automates hybrides, mais des semi-algorithmes sont connus lorsque la dynamique est suffisamment simple, comme une relation linéaire entre les dérivées. Avec la complexité croissante des systèmes actuels, ces modèles sont toutefois limités dans leur sémantique classique, pour modéliser des implémentations ou des systèmes dynamiques réalistes.

Dans cette thèse, nous étudions l'algorithmique de *sémantiques complexes* pour les automates temporisés et hybrides. D'une part, nous proposons des sémantiques implémentables pour les automates temporisés et nous étudions leur propriétés calculatoires: au contraire d'autres travaux, nous identifions notamment une sémantique qui est à la fois implémentable et qui a des propriétés de décidabilité. D'autre part, nous donnons des nouvelles approches algorithmiques pour l'analyse d'automates hybrides dont la dynamique est donnée par une fonction affine de ses variables.

Mots-clés: Temps-réel, Robustesse, Implémentabilité, Vérification, Automates temporisés, Automates hybrides.

Abstract

In the field of formal verification of real-time systems, major developments have been recorded in the last fifteen years. It is about logics, automata, process algebra, programming languages, etc. From the beginning, a formalism has played an important role: *timed automata* and their natural extension, *hybrid automata*. Those models allow the definition of real-time constraints using real-valued *clocks*, or more generally *analog variables* whose evolution is governed by differential equations. They generalize finite automata in that their semantics defines *timed words* where each symbol is associated with an occurrence timestamp.

The *decidability* and *algorithmic analysis* of timed and hybrid automata have been intensively studied in the literature. The central result for timed automata is that they are positively decidable. This is not the case for hybrid automata, but semi-algorithmic methods are known when the dynamics is relatively simple, namely a linear relation between the derivatives of the variables. With the increasing complexity of nowadays systems, those models are however limited in their classical semantics, for modelling realistic implementations or dynamical systems.

In this thesis, we study the algorithmics of *complex semantics* for timed and hybrid automata. On the one hand, we propose implementable semantics for timed automata and we study their computational properties: by contrast with other works, we identify a semantics that is implementable and that has decidable properties. On the other hand, we give new algorithmic approaches to the analysis of hybrid automata whose dynamics is given by an affine function of its variables.

Keywords: Real-Time, Robustness, Implementability, Verification, Timed automata, Hybrid automata.

Acknowledgments

Les effusions, dame, il déteste.
Selon lui, mettre en plein soleil
Son cœur ou son cul c'est pareil,
C'est un modeste.

Georges Brassens, *Le modeste*.

First of all, I would like to thank my thesis advisor Jean-François Raskin who has guided me throughout this thesis. For his continuous support of course, but also for his constant belief that we were on the good way and his enthusiastic view of research. Our discussions always gave rise to new ideas to explore. It was very stimulating to work with him. My thanks go also to Thierry Massart who introduced me to formal methods and research in computer science.

Jean-François has also given me the opportunity to meet Tom Henzinger in Brussels, and then to visit Tom twice in Lausanne. Tom is one of the leading researchers in the field of verification of real-time systems: I made great progress in doing research under his direction, and I really enjoyed to work with him. I would also like to thank the researchers and staff of MTC group who have welcomed me in Lausanne.

I wish to thank the other members of the jury, Gianluca Bontempi, Véronique Bruyère, Raymond Devillers, and François Laroussinie. A special thank goes to Raymond Devillers who accepted to carefully read a preliminary version of the manuscript. His comments and hints have significantly improved this thesis.

I gratefully acknowledge Martin De Wulf who worked with me during those last four years and has therefore continuously influenced this work, through the several interesting discussions we have had and the number of joint papers we have written (no less than six). I also thank Nicolas Markey who visited our research group as a post-doc in 2004 and contributed to essential results of this thesis.

I would not forget the whole Computer Science Department and the Verification Group at ULB, with a special mention to Gilles Geeraerts, a keyboard magician both as computer scientist and as organist.

I am grateful to the F.N.R.S. (the belgian National Fund for Scientific Research) for the support I received in the form of a research fellow position ("Aspirant du F.N.R.S.").

Finally, I thank all those who have here and there contributed to the achievement of this thesis.

Chronology

Much of Chapter 4 is based on material that was presented in the *Proceedings of the 7th International Workshop on Hybrid Systems, Computation and Control (HSCC)* in March 2004 [DDR04], and on an extended version of that paper that appeared in the international journal *Formal Aspects of Computing, Volume 17, Number 3* in October 2005 [DDR05a].

The main part of Chapter 5 appeared in the *Proceedings of the International Symposium of Formal Methods Europe* in July 2005 [DDR05b].

Chapter 6 is an extended version of a paper published in the *Proceedings of the Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)* in September 2004 [DDMR04].

A preliminary version of Chapter 7 was published in the *Proceedings of the 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)* in September 2005 [DHR05].

Aux hommes de bonne volonté.

Table of Contents

Chapter 1

Introduction

> As far as the laws of mathematics refer to reality, they are not certain;
> and as far as they are certain, they do not refer to reality.
>
> Albert Einstein.

1.1 Context

Embedded control systems take an increasingly important role in everyday life, from traffic light controllers to aircrafts and from telecommunication networks to medical systems. Many applications are *safety-critical* and their correctness is of course a highly desirable property. Embedded digital systems differ from classical computer systems in that they are reactive and real-time.

A *reactive* system interacts with an external environment, a physical system like a pump, an airplane or a railroad crossing. It receives inputs from the environment via sensors, and it should react through actuators to control the environment, that is essentially to maintain valid a set of properties. Usually, the execution of the system is not intended to terminate, and the control objective is to avoid ever reaching some set of error states (like maintaining a minimal distance between two aircrafts, or closing the barriers when the train is approaching).

A system is *real-time* if its correctness relies on the timing of actions and events. This is obviously the case when a real-time constraint is explicitly specified, *e.g.* a request must be responded within two time units. But it is also usual in other applications like the design of circuits and protocols where the relative delays between events (gate switch or message transmission) may affect the resulting behaviour.

The *model-based methodology* to develop such systems is a formal approach that can be decomposed in three main steps:

- The specification: a model of the environment under control, expressed in a formal language with a mathematical definition of its *semantics*. In addition, a control objective is provided, for example in the form of a set of bad states to avoid.

- The design: a model of a controller that describes a control strategy for steering the environment. Again, the model should be defined in mathematical terms.

- The verification: a procedure that establishes or refutes the correctness of the composition of the environment and the controller.

The *specification* is a manual task that is impossible to automate. It requires ingenuity and creative effort from the engineers and it may be long to obtain. The process can be eased by the use of design tools such as simulators. The expected properties of the model can also be verified in the third phase of the methodology.

The *design* is a task that is most often carried out manually, but there exist techniques to synthesize the controller, often related to game theory. No matter how it is obtained, the controller must be formally verified: either by a generic proof of correctness of the synthesis procedure, or by a specific proof for the application.

The *verification* consists in constructing such a proof, in general using automatic methods. Other approaches such as testing and simulation may be helpful to quickly detect errors, but they fail in proving the absence of errors by lack of an exhaustive analysis. By contrast, the techniques of *model-checking* on which we concentrate in this thesis aim at formally proving that the system is error free. Model-checking is difficult because non trivial real-time systems have an infinite number of states, an unbounded evolution in time and mixed discrete/continuous components. The presence of concurrency makes the task even harder.

We assume that a model of the environment is given using the formalism of *hybrid automata*, a formal language for dynamical systems with both discrete and continuous components. For the controller, we make a distinction between a model and an implementation. A model is a high-level description, classically in the same language as for the environment. Here, we choose *timed automata*, a subclass of hybrid automata. The model may exhibit non-deterministic choices and it has a mathematical semantics that is idealized. On the other hand, an implementation is also an abstract model, but its semantics should reflect much more accurately the behaviour of the physical realization of the controller. Like for the environment, the adequacy of the abstract model with regard to the real system cannot be checked formally. It can be justified by experiments and simulations. We argue that the implementation is not totally deterministic in its execution because the precise timing of a program is necessarily subject

to imprecisions, and cannot be uniquely predicted. However, the non-determinism related to the discrete choices is in general resolved in an implementation.

In this thesis, we concentrate on the verification of real-time reactive systems, described in terms of timed and hybrid automata, with a focus on open questions related to the analysis of *complex* semantics:

- For timed automata, we discuss the classical semantics and we show that it is not suitable for the implementation of controllers. We study a natural semantics that could be implemented, but we show that its verification is undecidable. Then, we propose a methodology to synthesize a correct implementation from a model, based on a new semantics that we show to be decidable.
- For hybrid automata, it is known that they are difficult to analyze, and that decidability is restricted to models having constant dynamics. However, there exist semi-algorithms (procedures that are sound but not guaranteed to terminate) to verify hybrid automata, but they are limited to relatively simple dynamics expressed by linear relations between the derivatives of the variables. We define an automatic technique to analyze more complex dynamics, where the derivatives are given as a linear function of the variables, and we show its applicability on a case study.

In the next two sections, we give more details about the motivations and goals of the thesis.

1.2 Verification of Robust Semantics for Timed Automata

As we have sketched above, in the model-based methodology to develop real-time systems, we have to establish the correctness of a system made up of:

- a hybrid automaton Env that models the environment, with a set Bad of bad states that are to be avoided.
- a timed automaton Cont that models a control strategy to prevent the environment to enter Bad.

In that context, the condition of correctness can be informally written as

$$\mathsf{Reach}(\llbracket \mathsf{Env} \rrbracket \parallel \llbracket \mathsf{Cont} \rrbracket) \cap \mathsf{Bad} = \varnothing \tag{1.1}$$

where $\mathsf{Reach}(\llbracket \mathsf{Env} \rrbracket \parallel \llbracket \mathsf{Cont} \rrbracket)$ denotes the set of states that are reached when the controller, embedded in the environment, is executed. Thus the system is correct if

none of the reachable states is a bad state (a formal definition of this notation is given in Chapter 2).

When the model of the controller has been proven correct, it would be valuable to ensure that an implementation Impl of that model can be obtained in a systematic way in order to ensure the *preservation of correctness*, that is to ensure that

$$\mathsf{Reach}([\![\mathsf{Env}]\!] \parallel [\![\mathsf{Impl}]\!]) \cap \mathsf{Bad} = \varnothing \tag{1.2}$$

holds by construction. This is often called *program refinement*: given a high-level description P_1 of a program, refine that description into another description P_2 such that the "important" properties of P_1 are maintained. Usually, P_2 is obtained from P_1 by reducing nondeterminism. The correctness of P_2 can be established using a notion of simulation (see Section 2.3 for a formal definition). Here, we have to show that the controller Cont simulates the implementation Impl, which means that Cont can mimic all the behaviours of Impl. In particular, this would imply that:

$$\mathsf{Reach}([\![\mathsf{Env}]\!] \parallel [\![\mathsf{Impl}]\!]) \subseteq \mathsf{Reach}([\![\mathsf{Env}]\!] \parallel [\![\mathsf{Cont}]\!]) \tag{1.3}$$

Then, the correctness of the implementation (Equation 1.2) would directly follow from the correctness of the model (Equation 1.1).

Unfortunately, this is often not possible in the context of real-time embedded controllers for several *fundamental* and/or *technical* reasons. First, the notion of time used in the traditional semantics of timed automata is *continuous* and defines *perfect clocks* (in the sense that their value is a real number, and their dynamics is of the form $\dot{x} = 1$), while implementations can access time only through *digital* and thus *finitely precise* clocks. Second, timed automata react *instantaneously* to events and time-outs. This is a convenient way to see reactivity and synchronization at the modeling level, but since implementations can only react within a given, usually small but not zero, *reaction delay*, any control strategy that relies for its correctness on that instantaneity cannot be implemented by any physical device no matter how fast it is. Third, timed automata may describe control strategies that are unrealistic, like *zeno* strategies or strategies that require the controller to act *faster and faster* [CHR02]. For those reasons, a naive implementation of a controller may not be correct, or the controller may not be implementable at all.

The infinite precision and instantaneity characteristics of the traditional semantics given to timed automata is very closely related to the *synchrony hypothesis* that is commonly adopted in the community of synchronous languages [Hal93, Ber00]. Roughly speaking, the synchrony hypothesis can be stated as follows:

> "*The program reacts to inputs of the environment by emitting outputs instantaneously.*"

The rationale behind the synchrony hypothesis is that the speed of a digital controller is usually so high with regard to the speed of the environment that the reaction time of the

controller can be neglected. This hypothesis *greatly simplifies* the view of the designer of an embedded controller: he/she does not have to take into account the performances of the platform on which the system will be implemented. This is coherent with the *model-based development methodology*. But like any hypothesis, the synchrony hypothesis should be *validated* not only by informal arguments, if we want to transfer correctness properties from models to implementations. To do so, we need a new semantics $[\![\mathsf{Cont}]\!]$ that realizes Equation 1.3.

Before going further, we discuss the *model of time*, an essential issue for modelling real-time systems: either the time domain is *discrete* like the natural numbers, or the time domain is *dense* like the rational or real numbers.

As far as we know, the models in physics use continuous time, except maybe at the sub-atomic scale. For the actual range of applications of embedded systems, the model of dense time is thus well suited. At the modelling level, we argue that it is an interesting notion of time because no assumption has to be made on the granularity of time which gives the most liberal framework for the design. This is desirable in the early phases of the development.

On the other hand, at the implementation level, the controller is executed on a *digital* hardware, with a discrete (tick-based) clock providing timing information with finite precision (or granularity). The controller has only access to discrete informations, essentially in the form of sequences of bits, both for its internal memory and for external input through its sensors and its clock. However, it is more convenient to keep a dense model of time because the controller is embedded in a physical environment, thus a continuous time device (sometimes the environment is even reduced to the sole model of time). Hence, the discrete (or sampled) behaviour of the controller is composed with the continuous-time behaviour of the environment, and thus the clocks of the controller are just observations of the environment's time which is taken as the reference. Those observations cannot be claimed to be *exact*, as they are given by clocks which are devices that *measure* and not define time. Therefore, a purely discrete-time controller would make the unrealistic assumption that the clocks are perfect, and that the behaviour of the program is *deterministic* with respect to time. We agree that in many situations the discrete time model is sufficient, because it is unlikely that an error occurs because of the violation of this assumption [Sta02, KMTY04]. However, a tiny probability does not mean impossibility, and it may happen that the slightest imperfection in clocks leads to totally different behaviours. We illustrate this more concretely with a simple example in Section 6.5.

After this discussion about time, we formulate two propositions for the semantics $[\![\mathsf{Cont}]\!]$. First, to model the imprecisions of the real system, we could relax the unrealistic constraint that the implementation has a sampling rate which is fixed and invariable. Instead, we ask that there exist some bounds on the sampling period. The controller can poll the inputs and compute new outputs every *between* α *and* β unit of time. This semantics appears as a good candidate for satisfying Equation 1.3. On the

computational side, we bring the following contribution:

- When α and β are fixed constants, this semantics is equivalent to a finite state system, and the condition of Equation 1.1 can be decided. However, the practitioner could object that when the constants are small, the model-checking would face a state explosion. On the other hand, the option to leave α and β as *parameters*, and then to synthesize a granularity such that the system is correct can be more appealing. Unfortunately, we show in Chapter 3 that this question is undecidable.

This result leads us to examine another implementable semantics, the Almost ASAP[1] semantics (AASAP-semantics), noted $[\![\mathsf{Cont}]\!]_{\Delta}^{\mathsf{AAsap}}$, where Δ is a parameter (see Section 4.4). This semantics was jointly defined with De Wulf and Raskin in [DDR04]. In this thesis, we concentrate on the computational aspects of the AASAP semantics. The detailed study of its properties will be part of the PhD thesis of Martin De Wulf.

The AASAP-semantics is a parametric semantics that leaves as a parameter the *reaction time* of the controller. This semantics relaxes the synchrony hypothesis in that it does not impose the controller to react instantaneously when a synchronization or a control action is enabled, but *within* Δ *time units*. The designer acts as if the synchrony hypothesis was true, *i.e.* he/she models the environment and the controller strategy without referring to the reaction delay. This reaction delay is taken into account during the *verification phase*: verifying the AASAP semantics of a controller, that is deciding whether:

$$\mathsf{Reach}([\![\mathsf{Env}]\!] \parallel [\![\mathsf{Cont}]\!]_{\Delta}^{\mathsf{AAsap}}) \cap \mathsf{Bad} = \varnothing \tag{1.4}$$

is roughly equivalent to check whether the control strategy is *robust* in the following sense:

> "*Is the strategy still correct if it is perturbed a little bit when executed on a device that has a finite speed and uses finitely precise clocks ?*"

To formally establish the implementability of the controller, we need to precisely define what *implementable* means. We choose to give a *program semantics* to real-time controllers, that intends to subsume (or simulate) any realistic implementation. We discuss our choice in Section 4.3.

The complete proofs of the formal link between models and implementations in that framework can be found in [DDR04, DDR05a]. On the computational side, we obtain the following results:

[1] ASAP = As soon as possible.

- Checking Equation 1.4 when Δ is a *fixed* rational number is decidable. This question corresponds to the problem of deciding if a controller can be correctly implemented on a given hardware, with known characteristics. The proof is detailed in Chapter 5. It uses an encoding of the AASAP semantics with timed automata. The result follows then from the decidability of reachability questions for timed automata. Thus, in practice the problem can be solved with the most efficient existing model-checkers for timed automata.

- When the hardware is not fixed, the problem is to decide the *existence* of $\Delta > 0$ such that Equation 1.4 holds. To solve the problem, we give two symbolic algorithms that improve the classical approach to parameter synthesis, either in a forward analysis or in a backward analysis. Those algorithms are correct, but not guaranteed to always terminate.

- In Chapter 6, we show that the question of existence has strong connections with robustness questions about timed automata, and that the problem is *decidable* when the model of the environment is a timed automaton that is also interpreted in a robust way, that is with perturbations bounded by Δ. This is not strictly equivalent to Equation 1.4, but still useful because at the same time we verify that the controller is implementable and that the model of the environment is robust, that is insensitive to an arbitrarily small level of noise. Indeed, if Equation 1.4 was falsified when Env is exposed to the slightest perturbations, we might conclude that the model of the system is flawed.

1.3 Verification of Affine Hybrid Automata

In the second part of this thesis, we shall be interested in the algorithmic analysis of complex semantics for hybrid automata, the formalism for modelling the environment. A hybrid automaton has a finite state structure that corresponds to the *modes* of the system, and in each mode, a set of continuous variables are evolving according to differential equations.

The algorithmic analysis of hybrid systems is theoretically limited by undecidability results [HKPV98]. The reachability questions that we have formulated in the previous sections are already undecidable when the dynamics of the environment is given by *rectangular inclusions* of the form $\dot{x} \in [a, b]$. On the practical side, there exists a reasonably efficient semi-algorithm to compute the set of reachable states [ACH+95], and when the semi-algorithm terminates, the correctness of the system is easily decided. However for more complex dynamics like affine dynamics of the form $\dot{x} = A\,x + B$, the computations are more difficult and only approximate methods are known.

We propose the following contributions:

- For the algorithmic analysis of affine dynamics hybrid automata, we automatically construct an over-approximation whose dynamics is given by rectangular inclusions. By over-approximation, we mean an automaton that simulates the affine dynamics automaton. This way, the approximation is sound for safety properties: if the correctness of the approximation can be established, we may directly conclude that the affine automaton is correct. Otherwise, we have to *refine* the approximation.
- Refinements are defined as approximations that are closer to the original system. That notion is precisely defined with the notion of simulation that we have already evoked in the previous section. We fix a natural measure of the imprecision of an approximation, and we define an optimization problem that aims at constructing the refinement that minimizes the imprecision. We give an algorithm to solve the problem for two dimensional systems (affine hybrid automata with two continuous variables). For the general case, we show how to obtain a refinement that is arbitrarily close to the optimal solution.
- We illustrate the practicability of the method on a benchmark case study. With a first prototype, we were able to verify quite efficiently a set of case studies.

1.4 Overview

The rest of the thesis is organized as follows.

Chapter 2 *Timed and Hybrid Automata*. We recall the definitions and properties of timed transition systems, an abstract formalism to model real-time systems. The notions of simulations and bisimulations are presented. Then, we review two concrete formalisms to denote real-time systems, timed and hybrid automata. For timed automata, we recall the region automaton construction to establish the decidability of reachability questions. For hybrid automata, the reachability questions being undecidable, we give a brief description of the semi-decision procedures used in practice.

Chapter 3 *Parametric Reasoning*. First, we review the existing results concerning parametric reachability questions for timed automata. Then we prove a new undecidability result for the class of *open timed automata*, where equality in timing constraints is forbidden. This result is of interest because open timed automata are *robust*, that is insensitive to arbitrarily small perturbations. Therefore, there may have been a chance that implementability of open timed automaton could be decided. Our result reduces that hope.

Chapter 4 *Implementability of Timed Automata*. We propose a novel approach to define implementability of timed automata. First, we propose a *program semantics* for timed automata that intends to be a realistic model of implementation,

as we take into account the most common sources of uncertainty in physical realization of real-time controllers: the precision of the clock, the reaction times, and the synchronization delays. The program semantics can be seen as a formal semantics for a *polling* procedure that periodically inspects the input sensors and computes new values for the output. The unique interest of this semantics is to be clearly implementable.

Second, we present the Almost ASAP semantics for timed automata. We introduce this parametric semantics to reason about the system at a more abstract level. The Almost ASAP semantics has the important characteristics that *faster is better*, a property that does not hold for the program semantics. More crucial is the link between the two semantics: the Almost ASAP semantics *simulates* the program semantics, and thus an implementation of a real-time controller will be correct by construction in our framework. Therefore, it is worth studying the verification problems of the next two chapters. The details and proofs of the properties of the Almost ASAP semantics are out of the scope of this thesis. The reader is referred to [DDR04, DDR05a].

Chapter 5 *Verification of the* AASAP *Semantics.* In this chapter, we first give a proof that the Almost ASAP semantics can be verified when its parameter is fixed to a rational constant. The proof is based on a careful study of the *urgency* in timed automata, and how to distribute this notion over a network of synchronizing timed automata. Beside the theoretical result, we give semi-algorithms to *synthesize* a value of the parameter in the AASAP semantics. Our procedures iteratively construct a simple necessary condition on the parameter, which becomes also a sufficient condition if the procedure terminates. The proof that the synthesis is algorithmically feasible is postponed to Chapter 6.

Chapter 6 *Robustness of Timed Automata.* Our goal in this chapter is to prove the decidability of the *implementability problem* for timed automata. Roughly, this problem asks, given a timed automaton A, whether there exists a perturbation of A (in the guards and/or the clock rates) such that some given error state is unreachable. This problem has an obvious connection with the synthesis problem for AASAP semantics, but also with a notion of *robustness* introduced by Puri [Pur98]: in a robust analysis of timed automata, Puri computes the states that are reachable when the automaton is perturbed by an arbitrarily small drift in the clocks rate. This set may be strictly greater than the classical reachable states. We complete these results with a detailed study of both guard and clock perturbations, which conducts to the decidability of the implementability problem.

On the other hand, we give a wide overview of the existing results about robustness of real-time and hybrid systems. We compare the approaches from the literature with our framework. Finally, we suggest an extension of our results to timed automata with diagonal constraints.

Chapter 7 *Verification of Affine Hybrid Automata.* We show how to systematically construct and refine rectangular abstractions of systems of linear differential equations. From a hybrid automaton whose dynamics is given by a system of affine differential equations, our method computes automatically a sequence of rectangular hybrid automata that are increasingly precise over-approximations of the original hybrid automaton. We prove an optimality criterion for successive refinements. We also show that this method can take into account a safety property to be verified, refining only relevant parts of the state space. The practicability of the method is illustrated on a benchmark case study.

Chapter 8 *Conclusion.*

Chapter 2

Timed and Hybrid Automata

> Que dites-vous ? C'est inutile ? Je le sais !
> Mais on ne se bat pas dans l'espoir du succès !
> Non! non! c'est bien plus beau lorsque c'est inutile !
>
> Edmond Rostand, *Cyrano de Bergerac*.

2.1 Preliminaries

Sets We denote by $\mathbb{R}$ the set of real numbers, by $\mathbb{Q}$ the rational numbers, by $\mathbb{Z}$ the integers, and by $\mathbb{N}$ the natural numbers (including 0). We use notations such as $\mathbb{R}^{\geq 0}$ for the positive real numbers and $\mathbb{R}^{>0}$ for the strictly positive real numbers, etc.

The *infimum* of a nonempty set $X \subseteq \mathbb{R}$ is the greatest lower bound of X and is denoted $\inf X$, with $\inf X = -\infty$ when X is unbounded from below. For $\sigma = \inf X$, we have $\forall x \in X : \sigma \leq x$ and $\forall \sigma' > \sigma \cdot \exists x \in X : x < \sigma'$. We define symmetrically $\sup X$, the *supremum* of X.

Norm, distance, neighbourhood For $p \in \mathbb{N}^{>0}$, we define the *p-norm* of a vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ by:

$$\|x\|_p = \sqrt[p]{|x_1|^p + \cdots + |x_n|^p}$$

For $p = 2$, $\|\cdot\|_2$ is the euclidean norm. Let $\|x\|_\infty = \max\{|x_1|, \ldots, |x_n|\}$. We use the *triangle inequality* in the following two equivalent forms. For all $x, y \in \mathbb{R}^n$, for all $p \in \mathbb{N}^{>0} \cup \{\infty\}$, we have:

$$\|x + y\|_p \leq \|x\|_p + \|y\|_p \qquad \|x - y\|_p \geq \|x\|_p - \|y\|_p$$

Given $p \in \mathbb{N}^{>0} \cup \{\infty\}$ and two vectors $x, y \in \mathbb{R}^n$, the *p-distance* between x and y is $d_p(x, y) = \|x - y\|_p$. Given two sets $X, Y \subseteq \mathbb{R}^n$, let $d_p(X, Y) = \inf\{d_p(x, y) \mid x \in$

$X \wedge y \in Y\}$. Given a set $X \subseteq \mathbb{R}^n$ and $\eta \in \mathbb{R}^{\geq 0}$, the *$\eta$-neighbourhood* of X is the set $\mathcal{N}_p(X,\eta) = \{x \in \mathbb{R}^n \mid \exists x' \in X : d_p(x,x') \leq \eta\}$. For $x \in \mathbb{R}^n$, define the shortcut $\mathcal{N}_p(x,\eta) = \mathcal{N}_p(\{x\},\eta)$ for the ball of radius η centered in x.

The *interior* of a set $X \subseteq \mathbb{R}^n$ is the set $\text{int}(X) = \{x \in \mathbb{R}^n \mid \exists \eta > 0 : \mathcal{N}_2(x,\eta) \subseteq X\}$ The set X is *open* iff $X = \text{int}(X)$. The *closure* of X is the set $(\overline{X}) = \{x \in \mathbb{R}^n \mid \forall \eta > 0 : \mathcal{N}_2(x,\eta) \cap X \neq \varnothing\}$. The set X is *closed* iff $X = \overline{X}$. The definition of open and closed sets is not changed if we use $\mathcal{N}_p(\cdot,\cdot)$ instead of $\mathcal{N}_2(\cdot,\cdot)$, for any $p \in \mathbb{N}^{>0} \cup \{\infty\}$.

A set $X \subseteq \mathbb{R}^n$ is *convex* if for all $x, y \in X$, for all $\lambda \in \mathbb{R}$ such that $0 \leq \lambda \leq 1$, we have $\lambda x + (1-\lambda)y \in X$.

Intervals An *interval* is a nonempty convex subset of the set $\mathbb{R}$ of real numbers. For a bounded interval I, we denote the left (resp. right) end-point of I by l_I (resp. r_I). For an interval I, we define $\mathsf{size}(I) = r_I - l_I$ if I is bounded, and $\mathsf{size}(I) = +\infty$ otherwise. Given two closed intervals I_1 and I_2, their *convex hull* is the closed interval $I_3 = I_1 \sqcup I_2$ such that $l_{I_3} = \min(l_{I_1}, l_{I_2})$ and $r_{I_3} = \max(r_{I_1}, r_{I_2})$. For bounded intervals with left end-point a and right end-point b, we use the notations $[a,b]$ for the closed interval, $]a,b[$ for the open interval, $[a,b[$ for the left-closed and right-open interval, and $]a,b]$ for the left-open and right-closed interval.

2.2 Timed Transition Systems

When the notions presented here are independent of the choice of a particular time model, we uniformly denote the time domain by $\mathbb{T}$. In the sequel, we use either the model of *continuous* time ($\mathbb{T} = \mathbb{R}^{\geq 0}$) or the model of *discrete* time ($\mathbb{T} = \mathbb{N}$). For *time-abstract* systems having a qualitative notion of time (either *some* time elapses or *no* time elapses) we use the singleton $\mathbb{T} = \{\mathsf{time}\}$. We introduce this time domain to deal with timed and untimed systems in the same framework. It is trivial to extend arithmetical operations for this time domain (the result is always time).

The executions of real-time systems can be represented by a state-transition graph called *timed transition system*, which is in general infinite. The vertices of the graph correspond to the state of the system (including timing informations), and the transitions correspond to a change in either the discrete or the continuous part of the state. Timed transition systems are an explicit formalism to define the executions of real-time system, and they cannot in general be effectively constructed because they are infinite. To specify real-time systems, we use high-level syntactical formalisms like timed and hybrid automata whose semantics is given in terms of timed transition systems. These formalisms are concise and, as we will see, very expressive.

Definition 2.1 A TTS (*timed transition system*) is a tuple $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$ where

- Q is a (possibly infinite) set of *states* ($\mathcal{T}$ is *finite* iff Q is finite),
- $Q_0 \subseteq Q$ is the set of *initial states*,
- $Q_f \subseteq Q$ is the set of *final states* (here considered as the *bad* states),
- Σ is a finite set of *labels*, including the *silent* label τ, and such that $\Sigma \cap \mathbb{T} = \varnothing$,
- and $\rightarrow \subseteq Q \times (\Sigma \cup \mathbb{T}) \times Q$ is the set of *transitions*.

□

We often write $q \xrightarrow{\sigma} q'$ as a shortcut for $(q, \sigma, q') \in \rightarrow$, and we call $q \xrightarrow{\sigma} q'$ a *discrete transition* if $\sigma \in \Sigma$, a *silent transition* if $\sigma = \tau$ and a *timed transition* if $\sigma \in \mathbb{T}$. When $\mathbb{T} = \{\mathsf{time}\}$, we say that the TTS is *time-abstract*.

For $k \in \mathbb{N}$, a *finite trajectory* of size k of a TTS $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$ is a finite sequence of the form:

$$\pi = (q_0, t_0)\, \sigma_1\, (q_1, t_1)\, \sigma_2 \ldots (q_{k-1}, t_{k-1})\, \sigma_k\, (q_k, t_k) \text{ such that}$$

- for all $0 \leq i \leq k$, we have $q_i \in Q$ and $t_i \in \mathbb{T}$,
- for all $1 \leq i \leq k$, $q_{i-1} \xrightarrow{\sigma_i} q_i$ and
  - either $\sigma_i \in \Sigma$ and $t_i = t_{i-1}$,
  - or $\sigma_i \in \mathbb{T}$ and $t_i = t_{i-1} + \sigma_i$.

Notice that only the differences $t_i - t_{i-1}$ are constrained in this definition, and so replacing each time stamps t_i by $t_i + \delta$ for $\delta \in \mathbb{T}$ yields another trajectory of $\mathcal{T}$. We write $|\pi| = k$ for the *size* (or the number of transitions) of π, and $\mathsf{Duration}(\pi) = t_k - t_0$ for the total *duration* of π.

The i-th state in π is written $\mathsf{state}_i(\pi) = q_i$ and we denote by $\mathsf{first}(\pi) = \mathsf{state}_0(\pi)$ (resp. $\mathsf{last}(\pi) = \mathsf{state}_{|\pi|}(\pi)$) the first (resp. last) state in π. We say that π is a trajectory *from* q_0 *to* q_k, and we sometimes write π more briefly as $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \ldots q_{k-1} \xrightarrow{\sigma_k} q_k$. The sequence of states $q_0\, q_1 \ldots q_k$ is called a *path* in $\mathcal{T}$, and the *trace* of π is the sequence:

$$\mathsf{trace}(\pi) = \lambda_1 \ldots \lambda_k \text{ with } \lambda_i = \begin{cases} \sigma_i & \text{if } \sigma_i \in \Sigma \\ \mathsf{time} & \text{otherwise.} \end{cases}$$

Let $\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_n}$ be the labels of π such that $\sigma_{i_j} \in \Sigma \setminus \{\tau\}$ and $i_1 < i_2 < \cdots < i_n$. The *timed word* defined by π is the sequence $\mathsf{word}(\pi) = (\sigma_{i_1}, t_{i_1} - t_0), (\sigma_{i_2}, t_{i_2} - t_0), \ldots, (\sigma_{i_n}, t_{i_n} - t_0)$.

We say that π is *stutter-free* if $trace(\pi)$ does not contain the silent label τ and we say that π is in *normal form* if $trace(\pi)$ does not contain the symbol time twice consecutively, that is, for each $1 \leq i < |\pi|$, either $\lambda_i \neq \mathsf{time}$ or $\lambda_{i+1} \neq \mathsf{time}$.

The *infinite trajectories* π_∞ of a TTS are defined in a similar way. In this case, we have $|\pi_\infty| = \infty$, $\mathsf{Duration}(\pi) = \lim_{k\to\infty} t_k - t_0$ and $\mathsf{last}(\pi_\infty)$ is undefined.

A TTS $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \to\rangle$ is said *normalized* if (i) for all timed transitions $(q,t,q') \in \to$ and $(q',t',q'') \in \to$, we have $(q, t+t', q'') \in \to$, and (ii) for all timed transitions $(q,t,q') \in \to$, for all t', $0 < t' < t$, we have $(q,t',q'') \in \to$ and $(q'', t-t', q') \in \to$ for some $q'' \in Q$. All the TTS we consider in the sequel are normalized.

Given a TTS $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \to\rangle$, the *untimed* version of $\mathcal{T}$ is the time-abstract TTS $\mathsf{Untime}(\mathcal{T}) = \langle Q, Q_0, Q_f, \Sigma, \to'\rangle$ where $\to' \subseteq Q \times (\Sigma \cup \{\mathsf{time}\}) \times Q$ is defined by:

$$(q,\sigma,q') \in \to' \text{ iff } \begin{cases} \sigma \in \Sigma \text{ and } (q,\sigma,q') \in \to \\ \text{or } \sigma = \mathsf{time} \text{ and } (q,t,q') \in \to \text{ for some } t \in \mathbb{T} \end{cases}$$

A state q of a TTS $\mathcal{T}$ is *reachable* if there exists a finite trajectory π of $\mathcal{T}$ such that $\mathsf{first}(\pi)$ is an initial state of $\mathcal{T}$ and $\mathsf{last}(\pi) = q$. The set of reachable states of $\mathcal{T}$ is noted $\mathsf{Reach}(\mathcal{T})$. For $A \subseteq Q$, we also use the notation $\mathsf{Reach}(\mathcal{T}, A)$ for the set of reachable states of the TTS $\langle Q, A, Q_f, \Sigma, \to\rangle$. Dually, a state q of $\mathcal{T}$ can *reach the final states* if there exists a trajectory π of $\mathcal{T}$ with $\mathsf{first}(\pi) = q$ and $\mathsf{last}(\pi)$ is a final state of $\mathcal{T}$. The set of states of $\mathcal{T}$ that can reach the final states is noted $\mathsf{Reach}^{-1}(\mathcal{T})$. Finally, let $\mathsf{Unsafe}(\mathcal{T}) = \mathsf{Reach}(\mathcal{T}) \cap \mathsf{Reach}^{-1}(\mathcal{T})$ be the set of reachable states that can reach the final states.

Definition 2.2 [Emptiness problem for TTS] A TTS $\mathcal{T}$ is *empty* if and only if $\mathsf{Unsafe}(\mathcal{T}) = \varnothing$. Given a TTS $\mathcal{T}$, the *emptiness problem for* TTSs asks whether $\mathcal{T}$ is empty. □

An equivalent formulation for the emptiness condition of $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \to\rangle$ is $\mathsf{Reach}(\mathcal{T}) \cap Q_f = \varnothing$. In this form, the emptiness condition is often referred to as a *safety* property. It can also be turned into an equivalent so-called *reachability* property as follows: $\mathcal{T}$ is empty iff $\mathsf{Reach}(\mathcal{T}) \subseteq Q_g$ where $Q_g = Q \backslash Q_f$ is the complement of Q_f.

We define the classical Pre and Post operator, leading to a more operational definition of the reachable states.

Given a set of states $A \subseteq Q$ in $\mathcal{T}$, the *one-step successors* of A is the set:

$$\mathsf{Post}_{\mathcal{T}}(A) = \{q' \in Q \mid \exists\, q \in A, \exists\, \sigma \in \Sigma \cup \mathbb{T} : q \xrightarrow{\sigma} q'\}$$

Similarly, the set of *one-step predecessors* of A in $\mathcal{T}$ is defined by:

$$\mathsf{Pre}_{\mathcal{T}}(A) = \{q \in Q \mid \exists\, q' \in A, \exists\, \sigma \in \Sigma \cup \mathbb{T} : q \xrightarrow{\sigma} q'\}$$

The reachable states of $\mathcal{T}$ can be written as the set $\mathsf{Reach}(\mathcal{T}) = \bigcup_{i\in\mathbb{N}} \mathsf{Post}^i_{\mathcal{T}}(Q_0)$ where $\mathsf{Post}^0_{\mathcal{T}}(A) = A$ and recursively $\mathsf{Post}^i_{\mathcal{T}}(A) = \mathsf{Post}_{\mathcal{T}}(\mathsf{Post}^{i-1}_{\mathcal{T}}(A))$ for $i \geq 1$. Symmetrically, we have $\mathsf{Reach}^{-1}(\mathcal{T}) = \bigcup_{i\in\mathbb{N}} \mathsf{Pre}^i_{\mathcal{T}}(Q_f)$ where $\mathsf{Pre}^0_{\mathcal{T}}(A) = A$ and recursively $\mathsf{Pre}^i_{\mathcal{T}}(A) = \mathsf{Pre}_{\mathcal{T}}(\mathsf{Pre}^{i-1}_{\mathcal{T}}(A))$ for $i \geq 1$.

2.3 Simulation and Bisimulations

Simulation relations are used to define a notion of refinement [Mil80]. A more abstract system is refined by a more concrete system if the former can simulate the latter.

Definition 2.3 [Simulation] Given two TTS $\mathcal{T}_1 = \langle Q^1, Q_0^1, Q_f^1, \Sigma^1, \rightarrow^1 \rangle$ and $\mathcal{T}_2 = \langle Q^2, Q_0^2, Q_f^2, \Sigma^2, \rightarrow^2 \rangle$, we write $\mathcal{T}_1 \succeq \mathcal{T}_2$ and say that $\mathcal{T}_1$ *simulates* $\mathcal{T}_2$ if $\Sigma^1 = \Sigma^2$ and there exists a relation $R \subseteq Q^1 \times Q^2$ such that:

1. for all $(q_1, q_2) \in R$, for each $\sigma \in \Sigma^2 \cup \mathbb{T}$, if $q_2 \xrightarrow{\sigma} q_2'$ then there exists $q_1' \in Q^1$ such that $q_1 \xrightarrow{\sigma} q_1'$ and $(q_1', q_2') \in R$,
2. for all $q_2 \in Q_0^2$, there exists $q_1 \in Q_0^1$ such that $(q_1, q_2) \in R$,
3. and for all $q_2 \in Q_f^2$, for all $q_1 \in Q^1$, if $(q_1, q_2) \in R$ then $q_1 \in Q_f^1$.

Such a witness relation R is called a *simulation relation* for $\mathcal{T}_1 \succeq \mathcal{T}_2$. □

For a relation $S \subseteq Q^1 \times Q^2$, let the inverse of S be the relation $S^{-1} = \{(q_2, q_1) \mid (q_1, q_2) \in S\}$.

Definition 2.4 [Bisimulation] Two TTS $\mathcal{T}_1$ and $\mathcal{T}_2$ are *bisimilar*, noted $\mathcal{T}_1 \approx \mathcal{T}_2$, if there exists a simulation relation R for $\mathcal{T}_1 \succeq \mathcal{T}_2$ and a simulation relation S for $\mathcal{T}_2 \succeq \mathcal{T}_1$ such that $R = S^{-1}$. □

Bisimulation is a stronger notion than mutual simulation in that two bisimilar TTS are mutually similar, but the converse does not hold.

We define a weaker version of simulation and bisimulation where the silent labels are hidden for external observations.

Given the TTS $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$, we define the *stutter-closed relation* $\twoheadrightarrow \subseteq Q \times (\Sigma \backslash \{\tau\} \cup \mathbb{T}) \times Q$ as follows: if $\sigma \in \Sigma \backslash \{\tau\}$ then $q \xrightarrow{\sigma}\!\!\!\!\!\twoheadrightarrow q'$ iff there exists a finite sequence $q_0, \ldots, q_k \in Q$ of states such that $q = q_0$, $q' = q_k$ and $q_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_i \xrightarrow{\sigma} q_{i+1} \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_k$; if $t \in \mathbb{T}$ then $q \xrightarrow{t}\!\!\!\!\!\twoheadrightarrow q'$ iff there exists a finite sequence $q_0, \ldots, q_{2k} \in Q$ of states and a finite sequence $t_0, \ldots, t_k \in \mathbb{T}$ of time stamps such that $q = q_0$ and $q_0 \xrightarrow{t_0} q_1 \xrightarrow{\tau} q_2 \xrightarrow{t_1} \ldots \xrightarrow{\tau} q_{2k} \xrightarrow{t_k} q'$ and $t = t_0 + \cdots + t_k$.

For the definition of weak simulation, we assume that in every sequence of silent transitions of the form $q \xrightarrow{\tau} \ldots \xrightarrow{\tau} q'$, if q' is final, then so is q. This ensures that we cannot "miss" a final state when we eliminate stuttering. Formally, we make the hypothesis that all the TTS $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$ that we consider are *visible*: if a state $q \notin Q_f$ and $q \xrightarrow{\tau} q'$, then $q' \notin Q_f$.

Definition 2.5 [Weak simulation] Given two TTS $\mathcal{T}_1 = \langle Q^1, Q_0^1, Q_f^1, \Sigma^1, \rightarrow^1 \rangle$ and $\mathcal{T}_2 = \langle Q^2, Q_0^2, Q_f^2, \Sigma^2, \rightarrow^2 \rangle$, we write $\mathcal{T}_1 \succeq_{weak} \mathcal{T}_2$ and say that $\mathcal{T}_1$ *weakly simulates* $\mathcal{T}_2$ if $\mathcal{T}_1' \succeq \mathcal{T}_2'$ where $\mathcal{T}_1' = \langle Q^1, Q_0^1, Q_f^1, \Sigma^1 \backslash \{\tau\}, \twoheadrightarrow^1 \rangle$ and $\mathcal{T}_2' = \langle Q^2, Q_0^2, Q_f^2, \Sigma^2 \backslash \{\tau\}, \twoheadrightarrow^2 \rangle$. A *weak simulation relation* for $\mathcal{T}_1 \succeq_{weak} \mathcal{T}_2$ is a simulation relation for $\mathcal{T}_1' \succeq \mathcal{T}_2'$. □

Definition 2.6 [Weak bisimulation] Two TTS $\mathcal{T}_1$ and $\mathcal{T}_2$ are *weakly bisimilar*, noted $\mathcal{T}_1 \approx_{weak} \mathcal{T}_2$, if there exists a weak simulation relation R for $\mathcal{T}_1 \succeq_{weak} \mathcal{T}_2$ and a weak simulation relation S for $\mathcal{T}_2 \succeq_{weak} \mathcal{T}_1$ such that $R = S^{-1}$. □

Using the results of [Par81], we have the following proposition.

Proposition 2.7 *For two* TTS $\mathcal{T}_1$ *and* $\mathcal{T}_2$*, if* $\mathcal{T}_1 \succeq \mathcal{T}_2$ *and* $\mathcal{T}_1$ *is empty, then* $\mathcal{T}_2$ *is empty.*

Note that for our purpose of checking emptiness, simulations and bisimulations are unnecessarily strong notions. Indeed, they can be used to establish more general classes of properties, like LTL and ∀CTL. However, simulations are useful as they provide a framework to present emptiness proofs in an inductive way.

Definition 2.8 [Hiding] Let $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$ be a TTS, and let $\Sigma' \subseteq \Sigma \setminus \{\tau\}$. The *hiding* of Σ' in $\mathcal{T}$ is the TTS $\mathcal{T}[\Sigma' := \tau] = \langle Q, Q_0, Q_f, \Sigma \backslash \Sigma', \rightarrow' \rangle$ where $(q, \sigma', q') \in \rightarrow'$ iff there exists $(q, \sigma, q') \in \rightarrow$ with either $\sigma \in \Sigma'$ and $\sigma' = \tau$ or $\sigma \notin \Sigma'$ and $\sigma' = \sigma$. □

2.4 Timed Automata

A timed automaton [AD94] is essentially a finite graph (with locations and edges) augmented with a set of clocks. Those clocks take values in the time domain $\mathbb{T}$ which can be either discrete ($\mathbb{T} = \mathbb{N}$) or continuous[1] ($\mathbb{T} = \mathbb{R}^{\geq 0}$). Invariants on locations and guards on edges are used to temporally constrain the system. The expressiveness and conciseness of the timed automata depends on the choice of the class of constraints allowed to define the invariants and the guards. Given a finite set of variables Var and a *class of predicates* $\mathcal{G}$, we denote by $\mathcal{G}(\mathsf{Var})$ the set of all predicates of the class $\mathcal{G}$ over the variables in Var. In the original model of timed automata, the constraints are rectangular.

Definition 2.9 [Rectangular predicates] Given a finite set of variables Var, a *closed rectangular predicate* over Var is a finite formula φ_c defined by the following grammar rule:

$$\varphi_c ::= \bot \mid \top \mid x \leq a \mid x \geq a \mid x = a \mid \varphi_c \wedge \varphi_c$$

[1] Other dense timed domains are sometimes considered in the literature, like the rational numbers $\mathbb{Q}^{\geq 0}$. Such choice would not modify the results presented.

where $x \in \mathsf{Var}$ and $a \in \mathbb{Q}$. An *open rectangular predicate* over Var is a finite formula φ_o defined by the following grammar rule:

$$\varphi_o ::= \bot \mid \top \mid x > a \mid x < a \mid \varphi_o \wedge \varphi_o$$

where $x \in \mathsf{Var}$ and $a \in \mathbb{Q}$.

We denote by $\mathsf{Rect_c}$ (resp. $\mathsf{Rect_o}$) the class of closed (resp. open) rectangular predicates. A *rectangular predicate* over Var is a formula φ defined by the grammar rule:

$$\varphi ::= \varphi_c \mid \varphi_o \mid \varphi_c \wedge \varphi_o$$

where $\varphi_c \in \mathsf{Rect}_c(\mathsf{Var})$ and $\varphi_o \in \mathsf{Rect}_o(\mathsf{Var})$. We denote by Rect the class of rectangular predicates. □

Definition 2.10 [Timed automaton] Given a class $\mathcal{G}$ of predicates, a *timed automaton* over $\mathcal{G}$ is a tuple $\langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ where

- Loc is a finite set of *locations* representing the discrete states of the automaton;
- Var is a finite set of real-valued variables, called *clocks*. Their first derivative with regard to time is equal to 1;
- $\mathsf{Init} : \mathsf{Loc} \to \mathcal{G}(\mathsf{Var})$ is the *initial condition*. The automaton can start in a location ℓ with values of its clocks satisfying $\mathsf{Init}(\ell)$;
- $\mathsf{Inv} : \mathsf{Loc} \to \mathcal{G}(\mathsf{Var})$ is the *invariant* condition. The automaton can stay in a location ℓ as long as the values of its clocks satisfy $\mathsf{Inv}(\ell)$;
- Lab is a finite alphabet of *labels* (including the *silent* label τ), corresponding to discrete events in the automaton;
- $\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Loc} \times \mathcal{G}(\mathsf{Var}) \times \mathsf{Lab} \times 2^{\mathsf{Var}}$ is a finite set of *edges*. An edge $(\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$ represents a jump from location ℓ to location ℓ' with a guard g, an event σ and a set $R \subseteq \mathsf{Var}$ of variables to be reset;
- $\mathsf{Final} : \mathsf{Loc} \to \mathcal{G}(\mathsf{Var})$ is the *final condition*, corresponding to the error states of the automaton.

□

An *Alur-Dill automaton* is a timed automaton over the class Rect. A *closed* (resp. *open*) *timed automaton* is a timed automaton over the class Rect_c (resp. Rect_o).

The semantics of a timed automaton A is given by a timed transition system $[\![A]\!]$. It can be defined in discrete time ($\mathbb{T} = \mathbb{N}$) or in continuous time ($\mathbb{T} = \mathbb{R}^{\geq 0}$). The *states* of $[\![A]\!]$ are pairs (ℓ, v) where $\ell \in \mathsf{Loc}$ and $v : \mathsf{Var} \to \mathbb{T}$ is a *valuation* over Var. If we fix

an order for the variables, we identify valuations and tuples in $\mathbb{T}^n$ where $n = |\mathsf{Var}|$. We often write v_i for the value $v(x_i)$ where x_i is the i-th variable in Var, and we also say that $v \in \mathbb{T}^n$. In order to specify the transition relation of $[\![A]\!]$, we must define when a predicate $\varphi \in \mathcal{G}(\mathsf{Var})$ is satisfied by a valuation v, noted $v \models^{\mathcal{G}} \varphi$ We denote by $[\![\varphi]\!]$ the set $\{v \mid v \models \varphi\}$ of valuations that satisfy the predicate φ. Below, we define $\models$ for the class of rectangular predicates.

Definition 2.11 Given a valuation $v : \mathsf{Var} \to \mathbb{T}$ and a predicate $\varphi \in \mathsf{Rect}(\mathsf{Var})$, we write $v \models \varphi$ and say that v *satisfies* φ if and only if (recursively):

- $\varphi \equiv \top$,
- or $\varphi \equiv x \bowtie a$ for $\bowtie \in \{<, \leq, =, \geq, >\}$ and $v(x) \bowtie a$,
- or $\varphi \equiv \varphi_1 \wedge \varphi_2$ and $v \models \varphi_1$ and $v \models \varphi_2$.

□

Addition and subtraction of valuations are defined as for vectors of $\mathbb{T}^n$ (by adding or subtracting their corresponding components). Sometimes, we use expressions such as $v+t$, $v-t$, $t-v$, etc. where $v \in \mathbb{T}^n$ and $t \in \mathbb{T}$. Those expressions are the valuations $v+\vec{t}$, $v-\vec{t}$ and $\vec{t}-v$ respectively, where $\vec{t} \in \mathbb{T}^n$ and $\vec{t}_i = t$ for $1 \leq i \leq n$. In particular, the valuation $v + t$ assigns the value $v(x) + t$ to each variable $x \in \mathsf{Var}$. On the other hand, $t \cdot v$ is a valuation that assigns the value $t \cdot v(x)$ to each variable $x \in \mathsf{Var}$.

Finally, given a set $R \subseteq \mathsf{Var}$, a valuation $v : \mathsf{Var} \to \mathbb{T}$ and a constant $c \in \mathbb{T}$, we write $v[R := c]$ for the valuation v' such that:

$$v'(x) = \begin{cases} c & \text{if } x \in R \\ v(x) & \text{if } x \in \mathsf{Var} \backslash R \end{cases}$$

We often use the shortcut $v[x := c]$ for $v[\{x\} := c]$.

Definition 2.12 Given a timed automaton $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ on a class $\mathcal{G}$ of predicates, the *semantics* of A is induced by the relation $\models^{\mathcal{G}}$. It is the TTS $[\![A]\!] = \langle Q, Q_0, Q_f, \Sigma, \to \rangle$ where :

- $Q = \{(\ell, v) \mid \ell \in \mathsf{Loc} \wedge v : \mathsf{Var} \to \mathbb{T} \wedge v \models^{\mathcal{G}} \mathsf{Inv}(\ell)\}$;
- $Q_0 = \{(\ell, v) \in Q \mid v \models^{\mathcal{G}} \mathsf{Init}(\ell)\}$;
- $Q_f = \{(\ell, v) \in Q \mid v \models^{\mathcal{G}} \mathsf{Final}(\ell)\}$;
- $\Sigma = \mathsf{Lab}$;
- The relation $\to \subseteq Q \times (\Sigma \cup \mathbb{T}) \times Q$ is defined as follows:

- Discrete transitions. For $\sigma \in \mathsf{Lab}$, $((\ell, v), \sigma, (\ell', v')) \in \rightarrow$ iff there exists an edge $(\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$ such that $v \models^{\mathcal{G}} g$ and $v' = v[R := 0]$.
- Timed transitions. For each $t \in \mathbb{T}$, $((\ell, v), t, (\ell', v')) \in \rightarrow$ iff $\ell' = \ell$, $v' = v + t$ and for every $t' \in [0, t] : v + t' \models^{\mathcal{G}} \mathsf{Inv}(\ell)$.

□

The emptiness problem for timed automata is a central question in the field of real-time systems verification.

Definition 2.13 [Emptiness problem for timed automata] Given a timed automaton A, the *emptiness problem for timed automata* asks whether $[\![A]\!]$ is empty. □

Another classical problem for timed automata is the *location reachability problem* where, given a location ℓ_f of a timed automaton A, it is asked to decide whether there exists a valuation v_f such that (l_f, v_f) is reachable in A. This problem is obviously reducible to emptiness, and the converse also holds since to transform an emptiness problem into an equivalent problem of location reachability, it suffices to create a new location ℓ_{err} and to add the edges $(\ell, \ell_{err}, g_\ell, \tau, \varnothing)$ with $g_\ell = \mathsf{Final}(\ell)$ for each location ℓ of the timed automaton.

Finally, notice that the invariants of Alur-Dill automata are not essential for the emptiness problem in the following sense: given an Alur-Dill automaton $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$, we can construct an Alur-Dill automaton A' with trivial invariants and such that A is empty if and only if A' is empty: $A' = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}', \mathsf{Inv}', \mathsf{Lab}, \mathsf{Edg}', \mathsf{Final}' \rangle$ such that:

- for all $\ell \in \mathsf{Loc}$, $\mathsf{Init}'(\ell) \equiv \mathsf{Init}(\ell) \wedge \mathsf{Inv}(\ell)$, $\mathsf{Inv}'(\ell) \equiv \top$ and $\mathsf{Final}'(\ell) \equiv \mathsf{Final}(\ell) \wedge \mathsf{Inv}(\ell)$, and
- $\mathsf{Edg}' = \{(\ell, \ell', g', \sigma, R) \mid (\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$ and $g' \equiv g \wedge \mathsf{Inv}(\ell) \wedge \mathsf{Inv}(\ell')[R := 0]\}$ where for a rectangular predicate φ and a set $R \subseteq \mathsf{Var}$, the predicate $\varphi[R := 0]$ is obtained by substituting each variable $x \in R$ by the constant 0 in φ, and interpreting the expressions $0 \bowtie a$ for $\bowtie \in \{<, \leq, =, \geq, >\}$ and $a \in \mathbb{Q}$ as expected.

Thus, invariants are useful at the design level to control time elapsing (by contrast with guards that control the timing of discrete actions) but for the verification of safety properties (or emptiness), they do not add expressive power. More generally, for a timed automaton on $\mathcal{G}$, this transformation holds if $\mathcal{G}$ is a class of predicates such that the set $[\![\varphi]\!]$ is convex for all formula $\varphi \in \mathcal{G}$ and $\mathcal{G}$ is closed under conjunction.

2.4.1 The Region Automaton

The emptiness problem has been shown decidable for Alur-Dill automata in [AD94]. The proof constructs the so-called *region automaton*, a finite time-abstract TTS, and shows that it is bisimilar to the untimed semantics of the original timed automaton, that is $R_A \approx \mathsf{Untime}(\llbracket A \rrbracket)$ where R_A is the region automaton of the timed automaton A. As we intensively use the region automaton in the sequel, we recall the construction and the main properties in continuous time ($\mathbb{T} = \mathbb{R}^{\geq 0}$). In discrete time, the construction is drastically simpler.

The construction of the region automaton assumes that the constants appearing in the predicates of the timed automaton (guards, invariants, initial and final conditions) are integers. This is not restrictive since emptiness is preserved when all the constants of an Alur-Dill automaton A are multiplied by a positive integer $c \in \mathbb{N}^{>0}$ (let A_c be the transformed automaton). More precisely, A is empty if and only if A_c is empty. To see this, observe that $\pi = (q_0, t_0)\,\sigma_1\,(q_1, t_1)\,\sigma_2 \ldots$ is a trajectory of $\llbracket A \rrbracket$ iff $\pi_c = (q'_0, c \cdot t_0)\,\sigma'_1\,(q'_1, c \cdot t_1)\,\sigma'_2 \ldots$ is a trajectory of $\llbracket A_c \rrbracket$ where $q'_i = (\ell_i, c \cdot v_i)$ if $q_i = (\ell_i, v_i)$, $\sigma'_i = c \cdot \sigma_i$ if $\sigma_i \in \mathbb{R}^{\geq 0}$ and $\sigma'_i = \sigma_i$ otherwise.

In Definition 2.14, we define an equivalence relation $\sim \subseteq [\mathsf{Var} \to \mathbb{R}^{\geq 0}] \times [\mathsf{Var} \to \mathbb{R}^{\geq 0}]$ between valuations over Var with the following properties:

- If $v \sim w$ then $\forall t_v \in \mathbb{R}^{\geq 0}$, $\exists t_w \in \mathbb{R}^{\geq 0} : v + t_v \sim w + t_w$;
- If $v \sim w$ then for all formulas $\varphi \in \mathsf{Rect}(\mathsf{Var})$ that appear in guards, invariants, initial and final conditions of A: $v \models \varphi$ iff $w \models \varphi$;
- If $v \sim w$ then $\forall R \subseteq \mathsf{Var} : v[R := 0] \sim w[R := 0]$.

The first condition ensures that two equivalent valuations visit the same equivalence classes of $\sim$ as time elapses. Informally, we say that $\sim$ is time-abstract invariant. The other two conditions ask that $\sim$ is invariant for the discrete transitions, that is two equivalent valuations satisfy exactly the same set of guards, and when some clocks are reset, the valuations remain equivalent.

For $a \in \mathbb{R}$, we write $\lfloor a \rfloor = \max\{k \in \mathbb{Z} \mid k \leq a\}$ to denote the integral part of a, and $\langle a \rangle = a - \lfloor a \rfloor$ to denote the fractional part of a.

Definition 2.14 [Region-equivalence] Let A be an Alur-Dill automaton with set of clocks Var. Let M be the largest constant appearing in the rectangular constraints (guards, invariants, initial and final conditions) of A. Two clock valuations $v, w : \mathsf{Var} \to \mathbb{R}^{\geq 0}$ are *region-equivalent* for A, noted $v \sim_A w$ if and only if

1. For all $x \in \mathsf{Var}$, if $v(x) \leq M$ or $w(x) \leq M$ then $\lfloor v(x) \rfloor = \lfloor w(x) \rfloor$;
2. For all $x, y \in \mathsf{Var}$, if $v(x) \leq M$ and $v(y) \leq M$ then $\langle v(x) \rangle \leq \langle v(y) \rangle$ iff $\langle w(x) \rangle \leq \langle w(y) \rangle$;

3. For all $x \in \mathsf{Var}$, if $v(x) \leq M$ then $\langle v(x) \rangle = 0$ iff $\langle w(x) \rangle = 0$;

A *region* of A is an equivalence class of $\sim_A$. We write $]v[$ for the region containing v and $\mathcal{R}_A$ for the set of all regions of A. □

Conditions (1) and (3) ensure that the same set of rectangular predicates of A are satisfied by two region-equivalent valuations v and w. Condition (2) imposes the same ordering of the fractional parts of clocks in v and w, so that one can determine the order in which clocks cross integers as time elapses, and thus which regions are the time successors of a given region. A corollary of conditions $(2)-(3)$ is that two region-equivalent valuations v and w such that $v(x) \leq M$ and $v(y) \leq M$ also satisfy the same set of *diagonal* constraints of the form $x - y \leq a$ or $x - y < a$ where $a \in \mathbb{N}$.

Obviously, the number of regions of A is finite. The number of regions depends only on the number n of clocks and the largest constant M of the automaton. We write this number $W(M, n)$ (or simply W when M and n are clear from the context).

Definition 2.15 [Operations on regions] A *time successor* of a region r is a region r' such that $\exists v \in r,\ \exists t \in \mathbb{R}^{\geq 0} : v + t \in r'$. Given a predicate $\varphi \in \mathsf{Rect}(\mathsf{Var})$, we say that a region r *satisfies* φ (noted $r \models \varphi$) if $\exists v \in r :\ v \models \varphi$. Given a set $R \subseteq \mathsf{Var}$, we write $r[R := 0]$ for the set $\{v[R := 0] \mid v \in r\}$. □

In Definition 2.15, if r is a region then the set $r[R := 0]$ is also a region.We are now ready to define the region automaton.

Definition 2.16 [Region automaton] Given an Alur-Dill automaton $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$, the *region automaton* of A is the finite time-abstract TTS $R_A = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$ where

- $Q = \{(\ell, r) \mid \ell \in \mathsf{Loc} \wedge r \in \mathcal{R}_A \wedge r \models \mathsf{Inv}(\ell)\}$;
- $Q_0 = \{(\ell, r) \in Q \mid r \models \mathsf{Init}(\ell)\}$;
- $Q_f = \{(\ell, r) \in Q \mid r \models \mathsf{Final}(\ell)\}$;
- $\Sigma = \mathsf{Lab}$;
- The relation $\rightarrow \subseteq Q \times (\Sigma \cup \{\mathsf{time}\}) \times Q$ is defined as follows:
  - Discrete transitions. For $\sigma \in \mathsf{Lab}$, $((\ell, r), \sigma, (\ell', r')) \in \rightarrow$ iff there exists an edge $(\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$ such that $r \models g$ and $r' = r[R := 0]$.
  - Timed transitions. We have $((\ell, r), \mathsf{time}, (\ell', r')) \in \rightarrow$ iff $\ell' = \ell$ and r' is a time successor of r.

□

The definition of the region automaton is very similar to the semantics of Alur-Dill automata (Definition 2.12) and in fact they are tightly coupled in the sense that the two sets $T_A = \{\mathsf{trace}(\pi) | \pi \text{ is a trajectory of } A\}$ and $T_{R_A} = \{\mathsf{trace}(\pi') | \pi' \text{ is a trajectory of } R_A\}$ are equal and therefore the emptiness problem for Alur-Dill automata is decidable.

Proposition 2.17 ([AD94]) *For the class of Alur-Dill automata, the emptiness problem is* PSPACE-COMPLETE.

In practice, the region automaton is not the most efficient way to algorithmically verify timed automata. The verification tools for timed automata (UPPAAL [LPY97], KRONOS [DOTY95], HYTECH [HHW95]) use symbolic representations of the state space and compute the reachable states by iterating the Post operator from the initial states [HNSY94]. We explain this approach in more details in the next section about hybrid automata, a generalization of timed automata. We would just mention an efficient data-structure that is specifically used to analyze timed automata: the difference bound matrices (DBM) [Dil89]. We give more details about them in Chapter 6. For the special case of closed timed automata, the reachability analysis can be made with another efficient data-structure: the binary decision diagrams (BDD) [Bry86]. The idea is that for deciding location reachability in a closed timed automaton, it is sufficient to consider integer-valued clocks [Bey01]. Therefore, both the location and the clock valuations can be encoded in the same BDD. The tool RABBIT implements this technique [BLN03].

2.4.2 Composition

The definition of the synchronized product of two timed automata over $\mathcal{G}$ requires that the class $\mathcal{G}$ is closed under conjunction: if φ_1 and φ_2 are predicates of $\mathcal{G}$, then so is the predicate $\varphi_1 \wedge \varphi_2$. All the classes of predicates we consider in the sequel are closed under conjunction.

Definition 2.18 [Synchronized product of timed automata] The *synchronized product* of two timed automata $A_1 = \langle \mathsf{Loc}^1, \mathsf{Var}^1, \mathsf{Init}^1, \mathsf{Inv}^1, \mathsf{Lab}^1, \mathsf{Edg}^1, \mathsf{Final}^1 \rangle$ and $A_2 = \langle \mathsf{Loc}^2, \mathsf{Var}^2, \mathsf{Init}^2, \mathsf{Inv}^2, \mathsf{Lab}^2, \mathsf{Edg}^2, \mathsf{Final}^2 \rangle$ on $\mathcal{G}$ is the timed automaton $A_1 \times A_2 = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ on $\mathcal{G}$ such that:

- $\mathsf{Loc} = \mathsf{Loc}^1 \times \mathsf{Loc}^2$;
- $\mathsf{Var} = \mathsf{Var}^1 \cup \mathsf{Var}^2$;
- $\mathsf{Init}(\ell_1, \ell_2) = \mathsf{Init}^1(\ell_1) \wedge \mathsf{Init}^2(\ell_2)$ for each $(\ell_1, \ell_2) \in \mathsf{Loc}$;
- $\mathsf{Inv}(\ell_1, \ell_2) = \mathsf{Inv}^1(\ell_1) \wedge \mathsf{Inv}^2(\ell_2)$ for each $(\ell_1, \ell_2) \in \mathsf{Loc}$;

- $\mathsf{Lab} = \mathsf{Lab}^1 \cup \mathsf{Lab}^2$;
- $((\ell_1, \ell_2), (\ell'_1, \ell'_2), g, \sigma, R) \in \mathsf{Edg}$ iff one of the following assertions holds:
  - $e_1 = (\ell_1, \ell'_1, g_1, \sigma, R_1) \in \mathsf{Edg}^1$, $e_2 = (\ell_2, \ell'_2, g_2, \sigma, R_2) \in \mathsf{Edg}^2$, $\sigma \neq \tau$, $g = g_1 \wedge g_2$ and $R = R_1 \cup R_2$.
  - $e_1 = (\ell_1, \ell'_1, g, \sigma, R) \in \mathsf{Edg}^1$, $\ell_2 = \ell'_2$ and $\sigma = \tau$ or $\sigma \notin \mathsf{Lab}^2$.
  - $\ell_1 = \ell'_1$, $\sigma = \tau$ or $\sigma \notin \mathsf{Lab}^1$, and $e_2 = (\ell_2, \ell'_2, g, \sigma, R) \in \mathsf{Edg}^2$.
- $\mathsf{Final}(\ell_1, \ell_2) = \mathsf{Final}^1(\ell_1) \wedge \mathsf{Final}^2(\ell_2)$ for each $(\ell_1, \ell_2) \in \mathsf{Loc}$;

□

In the product $A_1 \times A_2$, the timed automata A_1 and A_2 share common labels (in $\mathsf{Lab}^1 \cap \mathsf{Lab}^2$) and common variables (in $\mathsf{Var}^1 \cap \mathsf{Var}^2$). For technical reasons, we need both type of synchronizations in the sequel. However, when the synchronized product is used to design systems in a modular way, we often have no shared variables ($\mathsf{Var}^1 \cap \mathsf{Var}^2 = \varnothing$) and the same alphabet ($\mathsf{Lab}^1 = \mathsf{Lab}^2$). In that case, the set of timed words defined by trajectories of the composition $A_1 \times A_2$ is the intersection of the sets of timed words defined by trajectories of A_1 and A_2. Let $\mathsf{word}([\![A]\!]) = \{\mathsf{word}(\pi) \mid \pi \text{ is a trajectory of } [\![A]\!]\}$. Then, $\mathsf{word}([\![A_1 \times A_2]\!]) = \mathsf{word}([\![A_1]\!]) \cap \mathsf{word}([\![A_2]\!])$.

2.5 Hybrid Automata

Hybrid systems combine discrete evolutions (namely, mode changes and variable updates) and continuous evolutions through variables whose dynamics is governed by differential equations. The formalism of hybrid automata has been introduced to deal with such systems in a uniform way [Hen00]. The original definition is therefore very general. In this thesis, we focus on subclasses of particular interest. We have already presented timed automata, an important class of hybrid automata for which emptiness is decidable. When continuous variables are subject to rectangular flow constraints, that is constraints of the form $\dot{x} \in [a, b]$, hybrid automata are called *rectangular*. For that subclass of hybrid automata, there exists a reasonably efficient algorithm to compute the flow successor. Based on this algorithm, there exists an iterative method that computes the exact set of reachable states when it terminates. This semi-algorithm can be used to establish or refute *safety properties*. On the other hand, if the evolution of the continuous variables is subject to more complicated flow constraints, for example affine dynamics like $\dot{x} = 3x - y$, computing the flow successor is much more difficult and only approximate methods are known. We present in Chapter 7 a new approach to verify safety properties based on increasingly precise rectangular approximations of affine dynamics.

Predicates Let $X = \{x_1, \ldots, x_n\}$ be a finite set of variables. Given a valuation $v : X \to \mathbb{R}$ and $Y \subseteq X$, define $v_{|Y} : Y \to \mathbb{R}$ by $v_{|Y}(x) = v(x)$ for every $x \in Y$.

Definition 2.19 [Linear term] A *linear term* over a finite set of variables X is an expression of the form $y \equiv a_0 + \sum_{x_i \in X} a_i x_i$ where $a_i \in \mathbb{Q}$ $(0 \leq i \leq n)$ are rational constants. Given a valuation v over X, we write $[\![y]\!]_v$ for the real number $a_0 + \sum_{x_i \in X} a_i v(x_i)$. We denote by $\mathsf{LTerm}(X)$ the set of all linear terms over X. □

Definition 2.20 [Linear predicate] A *linear predicate* over X is a finite formula φ defined by the following grammar rule:

$$\varphi ::= y \bowtie 0 \mid \varphi \wedge \varphi$$

where $y \in \mathsf{LTerm}(X)$ and $\bowtie \in \{<, \leq, =, >, \geq\}$. The linear predicate is *closed* if $\bowtie \in \{\leq, =, \geq\}$.

We denote by Lin the class of linear predicates and by Lin_c the class of closed linear predicates. □

Definition 2.21 Given a valuation $v : X \to \mathbb{R}$ and a predicate $\varphi \in \mathsf{Lin}(\mathsf{Var})$, we write $v \models \varphi$ and say that v *satisfies* φ if and only if (recursively):

- $\varphi \equiv y \bowtie 0$ and $[\![y]\!]_v \bowtie 0$,
- or $\varphi \equiv \varphi_1 \wedge \varphi_2$ and $v \models \varphi_1$ and $v \models \varphi_2$.

□

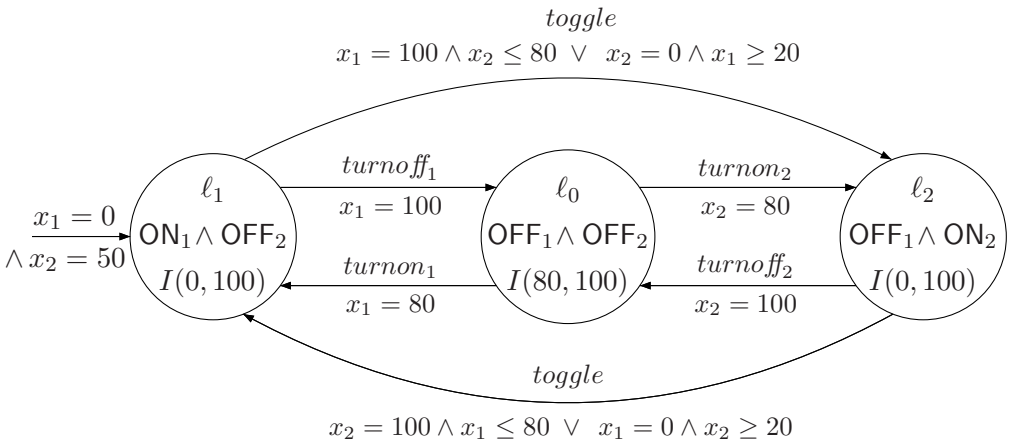
A *convex polyhedron* is a set $[\![p]\!] = \{v \mid v \models p\}$ defined by a linear predicate p, and a *polytope* is a convex polyhedron that is both closed and bounded.

An *affine dynamics predicate* over X is a formula of the form $\bigwedge_{x \in X} \dot{x} = t_x$ where $t_x \in \mathsf{LTerm}(X)$ $(x \in X)$ are linear terms over X and $\dot{x}$ represents the first derivative of x. Let $\dot{X} = \{\dot{x} \mid x \in X\}$. We denote by $\mathsf{Affine}(X, \dot{X})$ the set of all affine dynamics predicates over X. As usual, for an affine dynamics predicate p, we write $[\![p]\!]$ for the set of all valuations $v \in [X \cup \dot{X} \to \mathbb{R}]$ satisfying p.

We define three types of hybrid automata, with either affine, linear or rectangular dynamics [ACH+95, HKPV98].

Definition 2.22 [Hybrid automaton] A *hybrid automaton* H is a tuple $\langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ where:

- $\mathsf{Loc} = \{\ell_1, \ldots, \ell_m\}$ is a finite set of *locations*;



$\mathsf{ON}_1 \equiv \dot{x}_1 = h_1 - a_1 x_1 + b_1 x_2$
$\mathsf{OFF}_1 \equiv \dot{x}_1 = -a_1 x_1 + b_1 x_2$
$I(a, b) \equiv a \leq x_1 \leq b \wedge\ a \leq x_2 \leq b$

$\mathsf{ON}_2 \equiv \dot{x}_2 = h_2 - a_2 x_2 + b_2 x_1$
$\mathsf{OFF}_2 \equiv \dot{x}_2 = -a_2 x_2 + b_2 x_1$

Figure 2.1: A shared gas-burner.

- Lab is a finite set of *labels*, including the silent the *silent* label τ;
- $\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Lab} \times \mathsf{Loc}$ is a finite set of *edges*;
- $X = \{x_1, \ldots, x_n\}$ is a finite set of *variables*;
- $\mathsf{Init} : \mathsf{Loc} \to \mathsf{Lin}(X)$ gives the *initial condition* $\mathsf{Init}(\ell)$ of location ℓ. The automaton can start in ℓ with an initial valuation v lying in $[\![\mathsf{Init}(\ell)]\!]$;
- $\mathsf{Inv} : \mathsf{Loc} \to \mathsf{Lin}(X)$ gives the *invariant condition* $\mathsf{Inv}(\ell)$ of location ℓ. We require that $[\![\mathsf{Inv}(\ell)]\!]$ is bounded for every $\ell \in \mathsf{Loc}$. The automaton can stay in ℓ as long as the values of its variables lie in $[\![\mathsf{Inv}(\ell)]\!]$;
- Flow governs the evolution of the variables in each location:
  - either $\mathsf{Flow} : \mathsf{Loc} \to \mathsf{Affine}(X, \dot{X})$ and H is called an *affine* automaton,
  - or $\mathsf{Flow} : \mathsf{Loc} \to \mathsf{Lin}_c(\dot{X})$ and H is called a *linear* automaton;
  - or $\mathsf{Flow} : \mathsf{Loc} \to \mathsf{Rect}_\mathsf{c}(\dot{X})$ and H is called a *rectangular* automaton;

  We assume that $[\![\mathsf{Flow}(\ell)]\!] \neq \varnothing$ for every $\ell \in \mathsf{Loc}$.
- $\mathsf{Jump} : \mathsf{Edg} \to \mathsf{Lin}(X \cup X')$ with $X' = \{x'_1, \ldots, x'_n\}$ gives the *jump condition* $\mathsf{Jump}(e)$ of edge e. The variables in X' refer to the updated values of the variables after the edge has been traversed.

- $\mathsf{Final} : \mathsf{Loc} \to \mathsf{Lin}(X)$ gives the *final condition* $\mathsf{Final}(\ell)$ of location ℓ. In general, final conditions specify the unsafe states of the system.

□

Example Figure 2.1 represents an affine automaton modeling a single gas-burner that is shared for heating alternatively two water tanks. It has three locations ℓ_0, ℓ_1, ℓ_2 and two variables x_1 and x_2, the temperature in the two tanks. The gas-burner can be either switched off (in ℓ_0) or turned on heating one of the two tanks (in ℓ_1 or ℓ_2). The dynamics in each location is given by a combination of the predicates ON_i and OFF_i $(i = 1, 2)$ where the constants a_i model the heat exchange rate of the tank i with the room in which the tanks are located, b_i model the heat exchange rate between the two tanks and h_i depends on the power of the gas-burner. On every edge of the automaton, we have omitted the condition $x_1' = x_1 \wedge x_2' = x_2$ also written as $\mathit{stable}(x_1, x_2)$ that asks that the values of the variables are maintained when the edge is traversed. In the sequel, we fix the constants $h_1 = h_2 = 2$, $a_1 = a_2 = 0.01$ and $b_1 = b_2 = 0.005$.

Definition 2.23 [Semantics of hybrid automata] The *semantics* of an hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ is the TTS $[\![H]\!] = \langle S, S_0, S_f, \Sigma, \to \rangle$ where $S = \mathsf{Loc} \times \mathbb{R}^n$ is the *state space*, $S_0 = \{(\ell, v) \in S \mid v \in [\![\mathsf{Init}(\ell)]\!]\}$ is the *initial space*, $S_f = \{(\ell, v) \in S \mid v \in [\![\mathsf{Final}(\ell)]\!]\}$ is the *final space*, $\Sigma = \mathsf{Lab}$ and $\to$ contains all the tuples $((\ell, v), \sigma, (\ell', v'))$ such that:

- either there exists $e = (\ell, \sigma, \ell') \in \mathsf{Edg}$ such that $(v, v') \in [\![\mathsf{Jump}(e)]\!]$,
- or $\ell = \ell'$, $\sigma = \delta \in \mathbb{R}^{\geq 0}$ and there exists a continuously differentiable function $f : [0, \delta] \to [\![\mathsf{Inv}(\ell)]\!]$ such that $f(0) = v$, $f(\delta) = v'$ and for all $t \in [0, \delta]$:
  - either H is affine and $(f(t), \dot{f}(t)) \in [\![\mathsf{Flow}(\ell)]\!]$,
  - or H is linear or rectangular and $\dot{f}(t) \in [\![\mathsf{Flow}(\ell)]\!]$.

  Such a function f is called a *witness* for the transition $((\ell, v), \delta, (\ell', v'))$.

□

2.5.1 Algorithmic Verification

As for timed automata, we define an emptiness problem for hybrid automata.

Definition 2.24 [Emptiness problem for hybrid automata] Given an hybrid automaton H, the *emptiness problem for hybrid automata* asks whether $[\![H]\!]$ is empty. □

Unfortunately, the emptiness problem is undecidable already for rectangular automata.

Proposition 2.25 ([HKPV98]) *The emptiness problem is undecidable for both affine, linear and rectangular automata.*

We have already mentioned that emptiness is decidable for timed automata. A more detailed study of the boundary between decidability and undecidability is given in [HKPV98]. In particular, a decidability result is known for initialized rectangular automata. An hybrid automaton is *initialized* if for every edge $e = (\ell, \sigma, \ell')$ and for every variable x, if we have $\{v(\dot{x}) \mid v \in [\![\mathsf{Flow}(\ell)]\!]\} \neq \{v(\dot{x}) \mid v \in [\![\mathsf{Flow}(\ell')]\!]\}$ then the set $\mathsf{update}_e = \{w(x') \mid (v, w) \in [\![\mathsf{Jump}(e)]\!]\}$ does not depend on v. In words, whenever the flow condition changes for a variable by a discrete transition e, the variable is re-initialized to a new value in update_e that is independent of the ancient value.

Despite the undecidability of the general case, the emptiness problem has been attacked algorithmically: many of the existing tools (*a.o.* HyTech [HHW95], **d/dt** [ADMB00], PHAVer [Fre05]) use a symbolic analysis of the hybrid automaton with a forward and/or backward approach: starting from the initial (resp. unsafe) states, iterate the operator $\mathsf{Post}_{[\![H]\!]}$ (resp. $\mathsf{Pre}_{[\![H]\!]}$) until a fixed point is reached and then check emptiness of the intersection with the unsafe (resp. initial) states. By Proposition 2.25, those procedures are not guaranteed to terminate in general.

Remark The first versions of PHAVer that we have used do not implement the backward analysis. Nevertheless, for a rectangular automaton H it is possible to define the *reverse automaton* $-H$ such that $\mathsf{Pre}_{[\![H]\!]} = \mathsf{Post}_{[\![-H]\!]}$, so that $\mathsf{Reach}^{-1}_{[\![H]\!]} = \mathsf{Reach}_{[\![-H]\!]}$. Roughly, the construction consists in reversing the flow dynamics (an interval $[a, b]$ is replaced by the interval $[-b, -a]$) and the jump conditions (by permuting primed and unprimed variables). The other components are kept unchanged except the initial and unsafe sets which are swapped [HKPV98].

We briefly outline the principles of the algorithmic analysis of linear automata. Since the semantics of linear automata is infinite, the computation of $\mathsf{Post}_{[\![H]\!]}$ is done symbolically, with a symbolic representation of sets of states known as the *double description* for polyhedrons. A non-empty polyhedron $P \subseteq \mathbb{R}^n$ is represented by a pair $(\mathbf{A}, B)$ where $\mathbf{A}$ is a $m \times n$ matrix and B is a $m \times 1$ vector with rational coefficients where:

$$P = \left\{x \in \mathbb{R}^n \mid \mathbf{A}x^T \geq B\right\}$$

The pair $(\mathbf{A}, B)$ defines a systems of inequalities. We restrict this short presentation to topologically closed polyhedrons, defined by loose inequalities. Extension to strict inequalities is a technical issue discussed for example in [HPR94].

Operations such as inclusion tests and time successors are not easily handled when polyhedrons are represented by inequalities. Therefore, another representation has

been introduced: a closed polyhedron $P \subseteq \mathbb{R}^n$ can be represented by a pair (V, R), called the *generators* of P, where $V \subseteq \mathbb{R}^n$ is a finite set of *vertices* and $R \subseteq \mathbb{R}^n$ is a finite set of *rays*, with:

$$P = \left\{ \sum_{v_i \in V} \lambda_i \cdot v_i + \sum_{r_j \in R} \mu_j \cdot r_j \mid \lambda_i \geq 0, \mu_j \geq 0, \sum_i \lambda_i = 1 \right\}$$

Pair of rays r_i, r_j such that $r_i = -r_j$ are called *lines* and are often stored separately in implementations. The representation is also extended with *closure points* to deal with non-closed polyhedrons.

There exist algorithms for transforming a representation into the other, namely the Fourier-Motzkin procedure (or quantifier elimination) for computing the predicates representation from the generators [DE73, FR75], and the Chernikova's algorithm for computing the generators from a set of predicates [Che68].

We can now define the following operations. The *time successors* of a polyhedron P *under flow dynamics* given by $Q \neq \varnothing$ is the set:

$$P \nearrow Q = \{p + t \cdot q \mid p \in P, q \in Q, t \in \mathbb{R}^{\geq 0}\}$$

If (V, R) and (V', R') are the generators of P and Q respectively, then a set of generators for $P \nearrow Q$ is $(V, R \cup V' \cup R')$. Notice that if Q is a closed polyhedron, then $P \nearrow Q$ is a polyhedron, but this does not hold in general for non-closed polyhedrons Q.

The *discrete successors* of a polyhedron P by an edge $e = (\ell, \sigma, \ell')$ of a hybrid automaton H is the set:

$$\mathsf{post}_e(P) = \{v' \mid \exists v \in P : (v, v') \in [\![\mathsf{Jump}(e)]\!] \wedge v' \in [\![\mathsf{Inv}(\ell')]\!]\}$$

This set is easily computed (intersection and existential quantification are straightforward with formula representation).

The *inclusion test* $P \subseteq Q$ for a polyhedron P with generators (V, R) and a polyhedron $Q = \{x \mid \mathbf{A}x \geq B\}$ is equivalent to check that $\mathbf{A}v \geq B$ for each $v \in V$ and $\mathbf{A}r \geq 0$ for each $r \in R$. Finally, the *emptiness test* $P = \varnothing$ is equivalent to check that $V = \varnothing$.

Note that apart from the time successors, all the operations can be performed on the formula representation using quantifier elimination.

We can now sketch the semi-algorithm (shown as Algorithm 1) used to analyze linear hybrid systems. The states of the linear automaton H are represented by finite sets of pairs (ℓ, P) where $\ell \in \mathsf{Loc}$ and P is a polyhedron (stored using both the formula representation and the generators). The set of states corresponding to such a set R is $[\![R]\!] = \{(\ell, v) \mid \exists (\ell, P) \in R : v \in P\}$. If the linear automaton H is empty, the algorithm

computes the reachable states in variable R by iterating Post, without guarantee of termination. If H is non empty however, the procedure will eventually stop when a non empty intersection of R with the final states is detected.

Algorithm 1: Verification of linear hybrid systems: the VERIFY semi-algorithm.

Data : A linear automaton $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$.
Result : If H is empty then SAFE else UNSAFE.
begin
  $Bad \leftarrow \{(\ell, [\![\mathsf{Final}(\ell)]\!]) \mid \ell \in \mathsf{Loc}\}$;
  $R \leftarrow \{(\ell, [\![\mathsf{Init}(\ell)]\!]) \mid \ell \in \mathsf{Loc}\}$;
  $R_{old} \leftarrow \varnothing$;
  while $[\![R]\!] \not\subseteq [\![R_{old}]\!]$ **do**
    $R_{old} \leftarrow R$;
    $R \leftarrow \{(\ell, P \nearrow [\![\mathsf{Flow}(\ell)]\!]), (\ell', \mathsf{post}_e(P)) \mid (\ell, P) \in R \wedge e = (\ell, \sigma, \ell') \in \mathsf{Edg}\}$;
    if $[\![R]\!] \cap [\![Bad]\!] \neq \varnothing$ **then return** UNSAFE;
  return SAFE ;
end

A similar semi-algorithm implements the backward approach (by iterating Pre). Other approaches are possible such as mixed forward-backward where the forward and backward algorithms are executed in an interleaved fashion. All those variations are semi-algorithms since the problem is undecidable.

2.6 Conclusion

In this chapter, we have presented the timed and hybrid automata, the two main formalisms we study in this thesis. The central verification question about those formalisms is expressed by the emptiness problem, which is used to define reachability and safety properties.

This problem is decidable for timed automata (with the region automaton construction), but undecidable for the more general hybrid automata. However, semi-algorithmic verification of hybrid systems is possible, but without guarantee of termination.

In this chapter we have given an overview of the techniques that we are going to apply further:

- Simulations and bisimulations, a powerful tool to infer emptiness of a concrete systems from an emptiness proof of an abstract system;

- The region construction for timed automata, the basis for existing decidability results and further developments in this thesis;
- The semi-decision procedures for hybrid systems, that can also be applied for parametric verification of timed automata.

Chapter 3

Parametric Reasoning

> – Mais, Monsieur Fogg, ce laps de quatre-vingts jours n'est calculé que comme un minimum de temps !
>
> – Un minimum bien employé suffit à tout.
>
> – Mais pour ne pas le dépasser, il faut sauter mathématiquement des railways dans les paquebots, et des paquebots dans les chemins de fer !
>
> – Je sauterai mathématiquement.
>
> Jules Verne, *Le tour du monde en 80 jours*.

3.1 Introduction

In the model-based development methodology for real-time systems, we have to verify that the model (the composition of the environment and the controller) satisfies the control objective (here specified as a set of bad states of the environment to avoid). To do so, we can use the techniques that we have presented in Chapter 2 for timed and hybrid automata.

Our goal is also to ensure that the controller is implementable on a digital hardware. This means that we want to guarantee that the physical realization of the controller satisfies the control objective. However, to simplify the design process, the implementation details are often neglected or left as parameters in the model. Typically, for controllers that observe the environment at regularly spaced time instants, the sampling rate is not fixed *a priori*. In the verification phase, it is then asked to find a sampling rate such that the environment is prevented to enter the bad states.

More generally, the possibility to specify parametric timing constraints allows to design systems independently of a particular implementation. For example, the speed of the hardware or the time transmission in a communication protocol could be left as parameters in the early phases of the design. Then, when using the model for a

concrete application, the parameters are instantiated with a value satisfying a condition of correctness. However, it is well known that the problem of determining the existence of parameter values such that a parametric timed automaton satisfies a safety property is undecidable [AHV93, CHR02]. So the construction of a constraint that defines all the correct parameter values is often impossible.

The classical proofs of that result use the expressiveness of equality in timed automata to encode Turing machine's computations. The hope has then risen that decidability could be established for more robust models like open timed automata, that avoid equality constraints. In that case, we could specify that the sampling rate lies in an interval $]\alpha, \beta[$ with parametric bounds, and we would ask if there exist rational numbers α and β such that the bad states of the environment are avoided. A solution to that problem would give a realistic relaxed sampling semantics that could be implemented. Unfortunately, we shall show that this question cannot be answered and that parametric timed automata are thus "robustly" undecidable, as it is the case for classical timed systems [HR00].

In this chapter, we define the model of parametric timed automata in Section 3.2 and we review the known decidability and undecidability results for the reachability problem in parametric timed automata in Section 3.3. Then, we present a new proof of undecidability in continuous time for open timed automata that avoids equalities in clock constraints in Section 3.4. Therefore, this new result shows that parametric timed automata are quite robust as their ability to specify punctual events (through equality) is not the deep cause of their undecidability.

3.2 Parametric timed automata

In parametric timed automata (PTA for short), constants in guards and invariants can be replaced by parameters representing unfixed constants.

Definition 3.1 [Parametric rectangular predicates] Given a set Var of clocks and a set P of parameters, a *parametric rectangular predicate* over Var with parameters P is a finite formula φ defined by the following grammar rule:

$$\varphi ::= \bot \mid \top \mid x \bowtie a \mid \varphi \wedge \varphi$$

where $x \in \mathsf{Var}$, $a \in \mathbb{N} \cup \mathsf{P}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. We denote by $\mathsf{PRect}^{\mathsf{P}}$ the class of parametric rectangular predicates with parameters P. □

Given a valuation $v : \mathsf{Var} \to \mathbb{T}$, the truth value of a parametric predicate is parameterized by a valuation for the parameters. We assume the the parameters take their value in the set $\mathbb{T}$. This corresponds to the interesting problems studied in the literature. For continuous time ($\mathbb{T} = \mathbb{R}^{\geq 0}$), all the results presented hold if the parameters are valued in $\mathbb{Q}^{\geq 0}$.

Definition 3.2 Given a parameter valuation $\kappa : \mathsf{P} \to \mathbb{T}$, the relation $\models_\kappa$ is defined as follows. For a valuation $v : \mathsf{Var} \to \mathbb{T}$ and a parametric rectangular predicate $\varphi \in \mathsf{PRect}^{\mathsf{P}}$, we have $v \models_\kappa \varphi$ if and only if (recursively):

- $\varphi \equiv \top$,
- or $\varphi \equiv x \bowtie a$ for $\bowtie \in \{<, \leq, =, \geq, >\}$ and
  - either $a \in \mathbb{N}$ and $v(x) \bowtie a$,
  - or $a \in \mathsf{P}$ and $v(x) \bowtie \kappa(a)$;
- or $\varphi \equiv \varphi_1 \wedge \varphi_2$ and $v \models_\kappa \varphi_1$ and $v \models_\kappa \varphi_2$.

We denote by $[\![\varphi]\!]_\kappa$ the set $\{v \mid v \models_\kappa \varphi\}$. □

Definition 3.3 [PTA - Parametric Timed Automata] A *parametric timed automaton* is a pair $\langle A, \mathsf{P}\rangle$ where P is a finite set of parameters and A is a timed automaton on $\mathsf{PRect}^{\mathsf{P}}$. □

In the framework of hybrid automata, a parameter α can be seen as a variable that is not constrained by the predicate Init, that is never modified by any edge e (for any $(v, v') \in [\![\mathsf{Jump}(e)]\!]$ we have $v'(\alpha) = v(\alpha)$) and whose first derivative is 0 in every location ℓ (the constraint $\dot{\alpha} = 0$ appears in $\mathsf{Flow}(\ell)$).

Definition 3.4 Given a parameter valuation $\kappa : \mathsf{P} \to \mathbb{T}$, the *semantics* of a PTA $\langle A, \mathsf{P}\rangle$ is the semantics of A induced by the relation $\models_\kappa$, and is denoted $[\![A]\!]_\kappa$. □

Given a PTA A and a location ℓ_f, the set $\Gamma_{\ell_f}(A) = \{\kappa \mid (\ell_f, v_f) \in \mathsf{Reach}([\![A]\!]_\kappa)$ for some valuation $v_f\}$ contains all the parameter valuations such that ℓ_f is reachable in A.

Definition 3.5 Given a PTA $\langle A, \mathsf{P}\rangle$ and a location ℓ_f of A, the *parametric reachability problem* asks whether $\Gamma_{\ell_f}(A)$ is empty. □

The more general *synthesis problem* asks to compute a symbolic representation of the set $\Gamma_{\ell_f}(A)$, like a set of algebraic constraints. This symbolic representation should support boolean operations, projections and checking emptiness.

$\mathbb{T}$	Clocks compared to parameters	Other clocks	Parameters	
$\mathbb{N}$	1	any	any	$\surd$ [AHV93]
$\mathbb{R}$	1	0	any	$\surd$ [AHV93, Mil00]
$\mathbb{N}$ or $\mathbb{R}$	3	0	6	$\times$ [AHV93]
$\mathbb{R}$	3	0	1	$\times$ [Mil00]
$\mathbb{R}$	1	3	1	$\times$ [Mil00]

Table 3.1: Existing decidability ($\surd$) and undecidability ($\times$) results for PTA.

3.3 State of the Art

In Table 3.1, we give a summary of the existing results concerning the decidability of the parametric reachability problem, depending on the time domain and the number of clocks and parameters.

In [AHV93], the parametric reachability problem in discrete time is shown to be *decidable* for the class of PTA with an arbitrary number of clocks, but only one clock compared to parameters. The proof is in two steps. First they eliminate the non parametrically constrained clocks, and then they construct a formula of the additive theory of reals defining $\Gamma_{\ell_f}(A)$. Decidability of testing emptiness of $\Gamma_{\ell_f}(A)$ follows.

In continuous time, decidability is established only for PTA with only one clock, and the problem is NP-complete in this case [AHV93, Mil00]. However, for PTA with four clocks, the parametric reachability problem is undecidable even if only one clock is parametrically constrained [Mil00]. Finally, as stated by Theorem 3.6, the parametric reachability problem is undecidable in both discrete and continuous time for PTA with as few as three clocks and six parameters [AHV93].

Theorem 3.6 ([AHV93]) *The parametric reachability problem is undecidable in continuous time for general* PTA*.*

The proof is by reduction of the reachability problem for 2-counters machine which is known to be undecidable [Min67].

Definition 3.7 A *2-counters machine* is a tuple $\langle Q, q_0, q_m, \mathsf{Instr}\rangle$ where

- $Q = \{q_0, \ldots, q_m\}$ is a finite set of machine *states*;
- q_0 is the *initial* state;
- q_m is the *final* state;
- $\mathsf{Instr} : Q \to \mathcal{I}$ where $\mathcal{I}$ is the set of all *instructions* of the form:

- (increment) $C_k = C_k + 1$ **goto** q_i;
- (decrement) $C_k = C_k - 1$ **goto** q_i;
- (zero-testing) **if** $C_k = 0$ **then goto** q_i **else goto** q_j;

where $k \in \{1, 2\}$ and $q_i, q_j \in Q$ are machine states.

□

The decrement is not allowed if the counter value is 0. We may assume that a zero-testing is done before every decrement.

A *configuration* of a 2-counters machine $M = \langle Q, q_0, q_m, \mathsf{Instr} \rangle$ is a triple (q, c_1, c_2) where $q \in Q$ is a state of M and $c_1, c_2 \in \mathbb{N}$ are the values of the two counters C_1 and C_2. We write $\mathsf{Conf}_M = Q \times \mathbb{N} \times \mathbb{N}$ for the set of configurations of M.

Definition 3.8 An *execution* of a 2-counters machine M is an infinite sequence $\pi = \pi_0\pi_1 \ldots$ of configurations $\pi_i \in \mathsf{Conf}_M$ such that $\pi_0 = (q_0, 0, 0)$ and for all $i \geq 0$, if $\pi_i = (q, c_1, c_2)$ and $\pi_{i+1} = (q', c_1', c_2')$ then

- either $\mathsf{Instr}(q) \equiv C_k = C_k + 1$ **goto** q' and $c_k' = c_k + 1$ and $c_{3-k}' = c_{3-k}$,
- or $\mathsf{Instr}(q) \equiv C_k = C_k - 1$ **goto** q' and $c_k \neq 0$, $c_k' = c_k - 1$ and $c_{3-k}' = c_{3-k}$,
- or $\mathsf{Instr}(q) \equiv$ **if** $C_k = 0$ **then goto** q' **else goto** q'' and $c_k' = c_k = 0$ and $c_{3-k}' = c_{3-k}$,
- or $\mathsf{Instr}(q) \equiv$ **if** $C_k = 0$ **then goto** q'' **else goto** q' and $c_k' = c_k \neq 0$ and $c_{3-k}' = c_{3-k}$.

□

Definition 3.9 Given a 2-counters machine $M = \langle Q, q_0, q_m, \mathsf{Instr} \rangle$, the *reachability problem* is to decide if M has an execution containing a configuration (q_m, c_1, c_2) for some values c_1 and c_2. □

Proposition 3.10 ([Min67]) *The reachability problem for 2-counters machines is undecidable.*

The reduction presented in [AHV93] uses three clocks and six parameters to encode the value of the two counters, and the instructions of the 2-counters machine are translated into operations on clocks.

In [Mil00], an original proof is presented for continuous time. It works for PTA with three clocks and one parameter. The idea is to use a new undecidability result for irrational timed automata where constants can be irrational instead of integers. Once again, this result is proven by reduction of reachability problem for 2-counters

machines. Then, it is shown that the reduction is still applicable if the irrational constants are replaced by a parameter (since a rational parameter can be chosen arbitrarily close to an irrational). Refer to [Mil00] for details.

In the field of parametric real-time verification, there are several works where the parameters are introduced in real-time logics like TCTL [Wan95, WH97, ET99, AETP99, BDR03] or simultaneously in the model and in the logic [BR03]. The study of real-time logics is out of the scope of this thesis.

3.4 Robust Undecidability

In both undecidability proofs of the previous section (Theorem 3.6), the fact that PTA allow to define strong constraints of the form '$x = \alpha$' where α is a parameter is essential for simulating 2-counters machine. It could be argued that perhaps the parametric decision problem is undecidable simply because equality is too expressive. However, we now show that a 2-counters machine can be simulated without using equality. Our reduction technique is inspired by the widget construction presented in [CHR02].

A PTA $\langle A, \mathsf{P}\rangle$ is *open* if all the predicates (guards, invariants, initial and final conditions) of A are generated by the grammar:

$$\varphi ::= x < c \mid x > c \mid \varphi \wedge \varphi$$

where $x \in \mathsf{Var}_A$ and $c \in \mathbb{N} \cup \mathsf{P}$.

Theorem 3.11 *The parametric reachability problem is undecidable in continuous time for open* PTA *(with at least* 5 *clocks,* 2 *parameters and* 2 *of the clocks compared with parameters).*

Proof. Given a 2-counters machine $M = \langle Q, q_0, q_m, \mathsf{Instr}\rangle$, we construct an open PTA $\langle A_M, \{\alpha, \beta\}\rangle$ with five clocks and two parameters. Initially, all the clocks are equal to zero. The states $q_0, \ldots, q_m \in Q$ of the 2-counters machine are encoded by the locations $\ell_0, \ldots, \ell_m$ of A_M respectively. The construction is such that the location ℓ_m is reachable for some valuation of the parameters if and only if M reaches the state q_m. The value of each counter C_k is encoded by two clocks x_k and y_k of the timed automaton, and we use an additional clock t to generate pseudo-periodical *ticks*: the guard $t > \alpha$ appears on every edge, and the invariant $t < \beta$ appears on every location (except for the automaton of Figure 3.2 that we use in an initialization step). Also, we reset t on every edge so that a new tick occurs every between α and β time units with $\alpha < \beta$ (α and β are the parameters and typically their value is intended to be much less than 1).

After i such ticks, a clock x (initially 0) has a value in the interval $I_i =]i \cdot \alpha, i \cdot \beta[$. Now, assume that for some $n \in \mathbb{N} \setminus \{0\}$ (see also Figure 3.1):

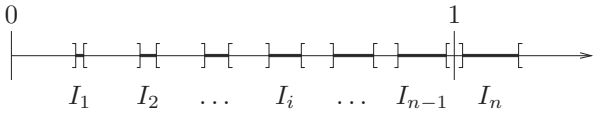


Figure 3.1: Intervals of the form $I_i =]i \cdot \alpha, i \cdot \beta[$ in which the clocks x and y are lying.

$(A1)$ the intervals I_i and I_{i+1} are disjoint for each $0 \leq i < n$;

$(A2)$ $I_{n-1} \subset [0,1]$ and $I_n \subset [1, +\infty[$.

Then, if we reset the clock x when $x \in I_n$, that is n ticks after the last reset, we can simulate a modulo-n counter. Now, we use the difference between two such counters to maintain the value of the machine counters as time elapses. Given a maximal constant n, we define the value c of a counter encoded with clocks x and y as follows:

$$\text{if } x \in I_i \text{ and } y \in I_j \text{ then } c = val(i,j) = \begin{cases} i - j & \text{if } i \geq j \\ n + i - j & \text{if } i < j \end{cases}$$

The assumption $(A1)$ guarantees the uniqueness of i such that $x \in I_i$ (and similarly for j). It is easy to establish the following invariance property for this encoding:

$$\forall 0 \leq i < n,\ \forall 0 \leq j < n :\ val(i+1 \bmod n, j+1 \bmod n) = val(i,j) \tag{3.1}$$

Note that the assumption $(A1)$ is equivalent to ask that I_{n-1} and I_n are disjoint, which is implied by $(A2)$. On the other hand, $(A2)$ is equivalent to the following condition on the parameters:

$$(n-1)\beta < 1 < n \cdot \alpha$$

We check this condition with the initialization widget A^{init} of Figure 3.2 whose location ℓ_0 (corresponding to the machine state q_0) is reachable if and only if there exists $n \in \mathbb{N} \setminus \{0\}$ such that $(n-1)\beta < 1 < n \cdot \alpha$. In fact n is the number of times the location a_0 is visited before reaching ℓ_0, thus the loop a_0, a_1 is taken $n-1$ times. Observe that the clock x_1 is always reset when $x_1 < \alpha$, so that when the edge (a_2, ℓ_0) is taken, we have $t < n \cdot \alpha$ and $t > 1$ which implies $1 < n \cdot \alpha$. On the other hand, the condition $(n-1)\beta < 1$ is trivially satisfied for $n = 1$. For $n > 1$, since y_1 is reset when $y_1 > \beta$, we have $t > (n-1)\beta$ in the location a_0 when the edge to a_2 is taken with $t < 1$. This entails $(n-1)\beta < 1$.

With this setting, the maximal value of the two counters is $n - 1 = \lfloor \frac{1}{\alpha} \rfloor = \lfloor \frac{1}{\beta} \rfloor$. Thus, with a lower value for α we can encode larger values of the counters. Since parameters are valued in $\mathbb{R}^{\geq 0}$, this is sufficient to guarantee a correct simulation of the 2-counters machine, if its counters remain bounded. If a counter overflow occurs

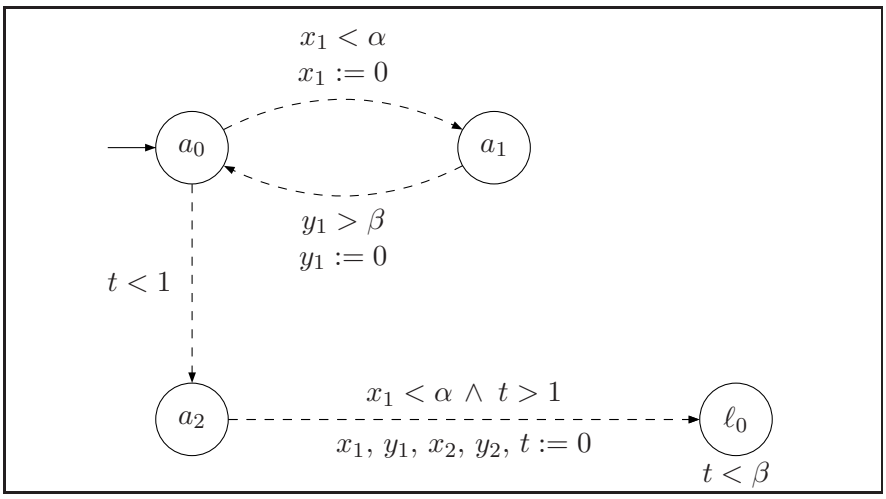


Figure 3.2: A^{init}

in the simulation of M, a special location is reached in A_M where it is impossible to reach ℓ_m. In summary,

- if M reaches q_m, then the values of its counters remain bounded, and so by choosing sufficiently small values for the parameters, A_M will be able to simulate M and thus to reach ℓ_m.
- On the other hand, if M does not reach q_m, then either the counters are unbounded and A_M falls in overflow whatever the choice of the value of the parameters, or A_M can mimic the execution of M forever (as before, by choosing sufficiently small values for the parameters), yet cannot reach ℓ_m since M does not reach q_m.

We present the widgets that we use to construct the timed automaton A_M. In all the subsequent figures, the invariant $t < \beta$ on each location, the guard $t > \alpha$ and the reset $t := 0$ on every edge are not depicted for the sake of clarity.

First, consider the idling automaton A_k^{idle} of Figure 3.4. This automaton maintains the value of the counter C_k by resetting x_k and y_k whenever they exceed 1. This widget is used to preserve the value of a counter while executing an instruction involving the other counter. The correctness of A_k^{idle} relies on Equation (3.1).

Now, we show how to execute the three types of instruction of M with A_M.

Increment An instruction of the form $\mathsf{Instr}(q_i) \equiv C_k = C_k + 1$ **goto** q_j is translated into the synchronized product $A_k^+ \times A_{3-k}^{idle}$ where A_k^+ is depicted in Figure 3.5. It is assumed that all their edges have the same label (not depicted) which ensures synchronizations of the two automata.

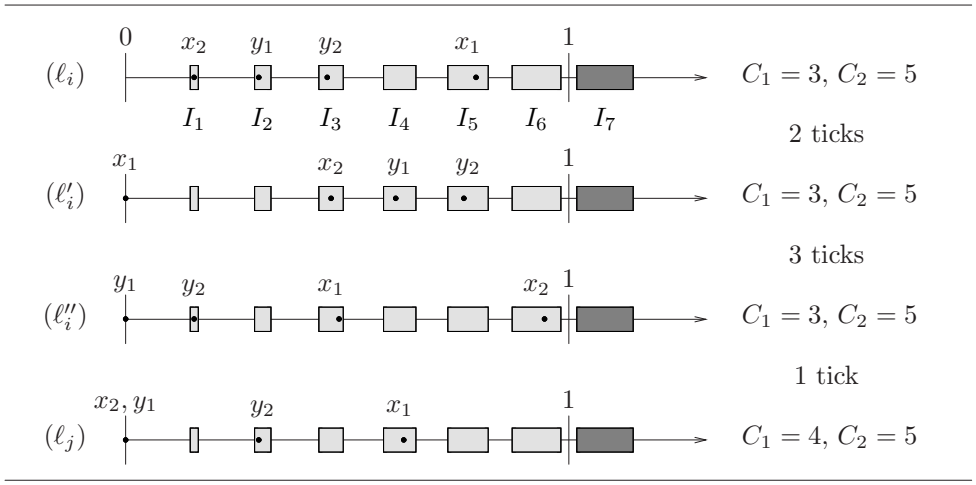


Figure 3.3: Incrementing the counter C_1.

We informally explain the structure of A_k^+. Remember that each edge is a tick. For the example, we have $n = 7$ and we increment C_1 (that is $k = 1$) starting from a configuration $(q_i, C_1 = 3, C_2 = 5)$ encoded by $x_1 \in I_5$, $y_1 \in I_2$, $x_2 \in I_1$ and $y_2 \in I_3$ as on Figure 3.3.

The first step is to obtain an encoding of C_1 such that $y_1 = 0$ in ℓ''_i. With a first tick, we cross the self-loop on ℓ_i and stay in location ℓ_i. After a second tick, x_1 breaks 1 so we jump to ℓ'_i and x_1 is reset. After three more ticks, we get into ℓ''_i with $y_1 = 0$ (note that y_2 has been reset meanwhile, due to the idling automaton). There is a direct jump to ℓ''_i without passing by ℓ'_i for the case where y_1 breaks 1 before x_1 does (or at the same tick). The second step is to increment C_1: we reset y_1 with the next tick and we proceed to ℓ_j. However, if x_1 was to break 1 at that time, it would mean that the counter overflows and that the simulation cannot continue. The deadlock location overflow is then reached.

Decrement A decrement of the form $\mathsf{Instr}(q_i) \equiv C_k = C_k - 1$ **goto** q_j is translated into the synchronized product $A_k^- \times A_{3-k}^{idle}$ where A_k^- is depicted in Figure 3.6. Again, all their edges have the same label.

Decrementing the counter C_k is the dual of incrementing: we proceed to location ℓ''_i (possibly via ℓ'_i) when x_k is reset so that $y_k \in I_{n-i}$ if the value of C_k is i. Then, with the next tick, we reset x_k so that $x_k \in I_0$ and $y_k \in I_{n-i+1}$. Thus, the value of C_k is now $i-1$ in ℓ_j. Note that there is no edge guarded by $x_k > 1 \wedge y_k > 1$ starting from location ℓ_i in Figure 3.6 since it corresponds to a counter equal to 0 which is prevented by the assumption that counters are zero-tested before decrementing.

Zero-testing An instruction of the form $\mathsf{Instr}(q_i) \equiv$ **if** $C_k = 0$ **then goto** q_j **else goto** $q_{j'}$ is translated into the synchronized product $A_k^0 \times A_{3-k}^{idle}$ where A_k^0 is depicted in Figure 3.7.

The value of a counter C_k is zero iff $x_k, y_k \in I_i$ for some i, which means that the two clocks will eventually exceed 1 during the same tick. This is checked by A_k^0, branching to either ℓ_j or $\ell_{j'}$.

The automaton A_M is now built up by concatenating A^{init} and each of the widget translated from the instructions of M. By concatenation, we mean taking the union of the locations and edges of the widgets, with initial location a_0 of A^{init} and final location ℓ_m. It is now clear from the above construction that the following claims are equivalent:

1. There exists a trajectory π of $[\![A_M]\!]_\kappa$ with $\mathsf{last}(\pi) = (\ell_m, v)$ for some valuation v.
2. There exists an execution π' of M containing a configuration (q_m, c_1, c_2) for some $c_1, c_2 \in \mathbb{N}$ and such that for all $i \geq 0$, if $\pi'_i = (q, c_1, c_2)$ then $c_1, c_2 \leq \lfloor \frac{1}{\kappa(\alpha)} \rfloor$.

This allows to conclude that $\Gamma_{\ell_m}(A_M)$ is not empty iff and only if the answer to the reachability problem for M is YES. From Proposition 3.10, this implies that the parametric reachability problem is undecidable for open PTA. ■

The proof that we have presented uses five clocks and two parameters, but three clocks are compared with parameters, namely x_1, y_1 and t. It is clear that the same reduction holds with only two clocks compared with parameters: in the initialization widget A^{init}, since all the clocks are reset before entering ℓ_0, we could swap for example x_1 and t.

3.5 Conclusion

We have seen that introducing parameters in the model of timed automata yields a parametric version of the emptiness problem that is undecidable in continuous time except for some restricted cases, with few interactions with parameters (and only one clock in continuous time). Those decidable classes are of limited usefulness.

We have strengthened this result with a new proof of undecidability for parametric timed automata. Unlike the classical proofs of undecidability, our proof does not rely on the use of equality in timing constraints and thus it is more robust. Formally, it applies to *open* parametric timed automata, with at least two parameters and five clocks (among which two are compared with the parameters). It is an open question whether this number of clocks and parameters is tight for undecidability.

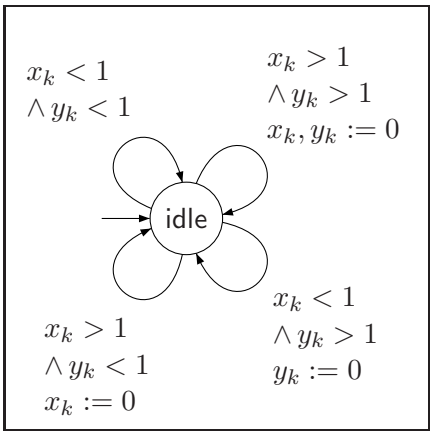


Figure 3.4: Idling with A_k^{idle}.

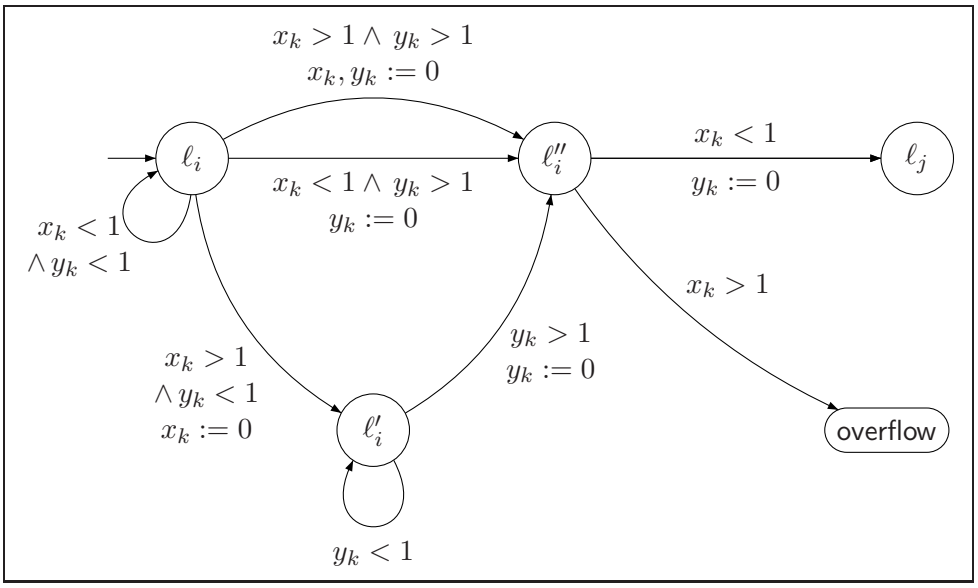


Figure 3.5: Incrementing C_k with A_k^+.

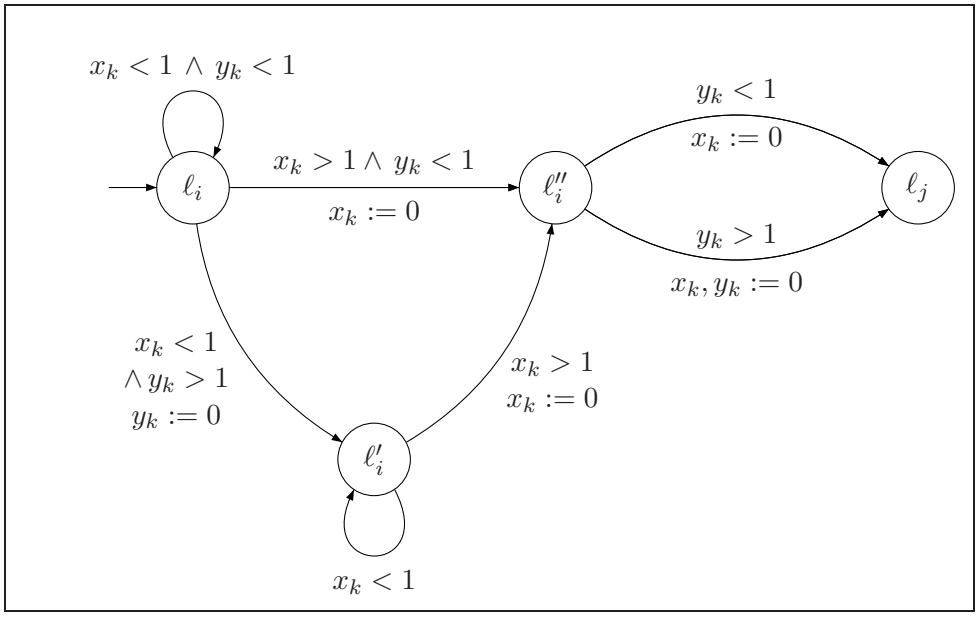


Figure 3.6: Decrementing C_k with A_k^-.

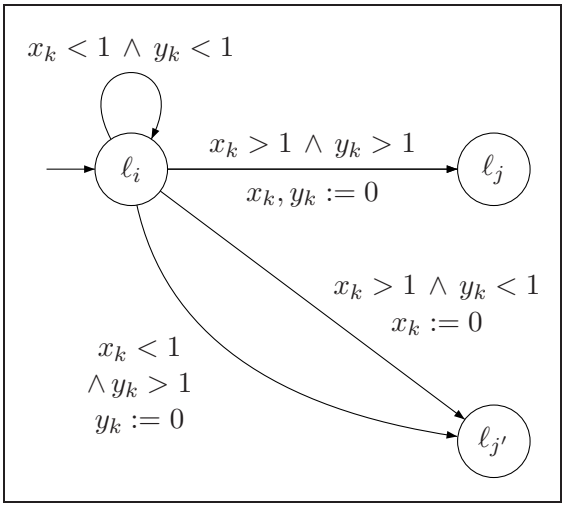


Figure 3.7: Zero-testing of C_k with A_k^0.

Since timed automata are quite expressive, those undecidability results for the parametric reachability problem are not surprising. In Chapter 6 however, we identify a class of PTA for which we establish decidability and that has an interest in practice. This class has *one* parameter ($\mathsf{P} = \{\Delta\}$) and allows clock constraints of the form $\varphi ::= x \geq a - \Delta \mid x \leq a + \Delta \mid \varphi \wedge \varphi$ with $a \in \mathbb{N}$. The idea is to enlarge the classical constraints of timed automata by an unfixed value Δ to model the imprecisions of the implementation. The details are given in Chapter 4.

Chapter 4

Implementability of Timed Automata

> Loi de Hofstadter: tout prend toujours plus de temps que prévu, même en tenant compte de la Loi de Hofstadter.
>
> Douglas Hofstadter, *Gödel, Escher et Bach, les brins d'une guirlande éternelle*.

4.1 Introduction

Formal methods are now widely used for the specification and verification of computer systems, and in particular real-time systems. The verification techniques apply to mathematical models because the formal arguments hold only in a well defined context. The need for a rigorous approach to the design of real-time systems is nowadays understood by both the research community and industry.

It is important to remind that verification gives guarantees for the correctness of a *model* which is a more or less sound description of a physical realization. Of course, when the model is given in terms of the formal semantics of a programming language, the designer can be more confident that the implementation of the model is faithful and thus correct.

However, embedded real-time controllers are seldom directly developed in the form of a compilable/executable program. As for traditional programs, the development of real-time reactive systems often follows a cyclic design methodology, like the model-based methodology. In the early phases of the design, it is more convenient for the engineers to take a high-level view of the conception, not considering the implementation details. At those steps, the verification techniques have proven useful for eliminating the mistakes. Later, the models are refined and transformed to be implemented on a real-time platform. At the level of code, it is often more difficult to formally check correctness because the system is much more detailed. If the implementation is

automatically generated, a proof that the synthesis procedure preserves correctness is sufficient, and this is more appealing

In this chapter, we present the tools that enable us to apply this schema to the design of real-time reactive controllers. In Section 4.2, we define structuration and composition of timed systems. In Section 4.3 we discuss a formal model for the implementation of real-time embedded controllers, called the *program semantics*. We argue that this model is quite general, and that many usual real-time platforms fit the definition. Therefore it is a convincing model for implementations. Unfortunately, we will see that the program semantics is not well suited for verification. In Section 4.4, we present a new semantics for timed automata, the Almost ASAP semantics, whose purpose is to serve in the verification phase. This semantics has nice properties that are useful for the verification and there is a formal connection with the program semantics. Finally in Section 4.5, we give an overview of related works where similar problems are addressed.

The algorithmic analysis of the Almost ASAP semantics is presented in Chapters 5 and 6.

4.2 Structured Timed Systems

We use structured systems to model *open* systems where the set of synchronization labels is partitioned into *uncontrollable* (namely the input events released by the environment), and *controllable*, corresponding to either output events visible to the environment or internal events for internal changes.

4.2.1 Structured Timed Transition Systems

The following definition refines Definition 2.1.

Definition 4.1 A STTS (*structured timed transition system*) is a tuple $\mathcal{S} = \langle Q, Q_0, Q_f, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_\tau, \rightarrow \rangle$ such that Σ_{in}, Σ_{out}, Σ_τ is a partition of $\Sigma = \Sigma_{\mathsf{in}} \cup \Sigma_{\mathsf{out}} \cup \Sigma_\tau$ and $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$ is a TTS. We say that $\mathcal{S}$ is given by the TTS $\mathcal{T}$ *structured* by $(\Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_\tau)$. We assume that $\tau \in \Sigma_\tau$. □

A classical notion of refinement is induced by simulations: a TTS $\mathcal{T}_1$ refines a TTS $\mathcal{T}_2$ if $\mathcal{T}_2 \succeq \mathcal{T}_1$. As mentioned in Section 2.3, refinements preserve safety properties (or emptiness), so that the correctness of $\mathcal{T}_2$ implies the correctness of $\mathcal{T}_1$. However, we do no want the relation $\mathcal{T}_2 \succeq \mathcal{T}_1$ to hold trivially because $\mathcal{T}_1$ might be blocking at some point. Classically, it is asked that $\mathcal{T}_1$ is non-blocking: it must accept every input in every state and it may not prevent time to diverge. The first condition is known as input enabledness [LT87], and the second as non-zenoness [AL94].

Definition 4.2 [Input enabled STTS] A STTS $\mathcal{S}$ given by $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$ structured by $(\Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_\tau)$ is *input enabled* if for all $\sigma \in \Sigma_{\mathsf{in}}$, for all $q \in Q$ there exists $q' \in Q$ such that $(q, \sigma, q') \in \rightarrow$. □

Input enabledness is easy to verify and is often obtained by construction. In some cases however, we have not input enabledness, for example when we want to model a process that terminates; then, we can always add the missing transitions from blocking states going to an error state contained in the final set of the TTS.

Definition 4.3 [Non-zeno STTS] A STTS $\mathcal{S}$ given by $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$ structured by $(\Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_\tau)$ is *non-zeno* if for all finite trajectory π of $\mathcal{T}$ with $\mathsf{first}(\pi) \in Q_0$, there exists a trajectory π' of $\mathcal{T}$ such that $\mathsf{first}(\pi') = \mathsf{last}(\pi)$, $\mathsf{trace}(\pi') \cap \Sigma_{\mathsf{in}} = \varnothing$ and $\mathsf{Duration}(\pi') = 1$. □

Non-zenoness is a liveness assumption [AS85] and it is the only liveness condition we need in practice because under this condition, the important liveness properties become safety properties [Hen92]. For example, the bounded response property, stating that every request must be followed by a grant within some delay δ, is a liveness property. But if time diverges then we can equivalently say that an error state is reached whenever a request is followed by a delay δ without a grant has occurred. Algorithmic aspects of checking non-zenoness are presented in [HNSY94].

The non-zenoness condition rules out systems issuing an infinite number of actions in a finite amount of time, for example at times $\frac{1}{2}$, $\frac{3}{4}$, $\frac{7}{8}$, $\frac{15}{16}$, etc. Such systems are clearly not physically realizable. However it is not sufficient that a system is non-zeno to be implementable [CHR02]. Consider the following sequence of time stamps:

$$\rho = 0, \frac{1}{2}, 1, 1\frac{1}{4}, 2, 2\frac{1}{8}, 3, 3\frac{1}{16}, \ldots$$

where $\rho_{2i} = i$ and $\rho_{2i+1} = i + \frac{1}{2^{i+1}}$ $(i \geq 0)$. Although ρ is diverging, it is impossible for a hardware to issue actions at times ρ_i, because the time distance between two successive actions is not bounded from below. To avoid such unwanted behaviours, we should impose a more realistic condition that two successive actions can always be separated by at least a fixed amount of time $\delta > 0$.

Definition 4.4 [Composition of STTS] Two STTS $\mathcal{S}_1$ given by $\langle Q^1, Q^1_0, Q^1_f, \Sigma^1, \rightarrow^1 \rangle$ structured by $(\Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau)$ and $\mathcal{S}_2$ given by $\langle Q^2, Q^2_0, Q^2_f, \Sigma^2, \rightarrow^2 \rangle$ structured by $(\Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau)$ are *composable* if for every $\sigma \in \Sigma^1 \cap \Sigma^2 \setminus \{\tau\}$, we have either $\sigma \in \Sigma^1_{\mathsf{in}} \cap \Sigma^2_{\mathsf{out}}$ or $\sigma \in \Sigma^2_{\mathsf{in}} \cap \Sigma^1_{\mathsf{out}}$.

Their composition, noted $\mathcal{S}_1 \| \mathcal{S}_2$ is the STTS given by $\mathcal{T}$ structured by $(\Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_\tau)$ where:

- $\Sigma_{\mathsf{in}} = (\Sigma^1_{\mathsf{in}} \cup \Sigma^2_{\mathsf{in}}) \setminus (\Sigma^1 \cap \Sigma^2)$;
- $\Sigma_{\mathsf{out}} = (\Sigma^1_{\mathsf{out}} \cup \Sigma^2_{\mathsf{out}}) \setminus (\Sigma^1 \cap \Sigma^2)$;
- $\Sigma_\tau = (\Sigma^1_\tau \cup \Sigma^2_\tau) \cup (\Sigma^1 \cap \Sigma^2)$;

and $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$ such that $Q = Q^1 \times Q^2$, $Q_0 = Q^1_0 \times Q^2_0$, $Q_f = \{(q_1, q_2) \in Q \mid q_1 \in Q^1_f \vee q_2 \in Q^2_f\}$, $\Sigma = \Sigma_{\mathsf{in}} \cup \Sigma_{\mathsf{out}} \cup \Sigma_\tau$, and $((q_1, q_2), \sigma, (q'_1, q'_2)) \in \rightarrow$ iff one of the following three assertions holds:

- $\sigma \in (\Sigma^1 \cap \Sigma^2 \setminus \{\tau\}) \cup \mathbb{R}^{\geq 0}$ and $(q_1, \sigma, q'_1) \in \rightarrow_1$ and $(q_2, \sigma, q'_2) \in \rightarrow_2$
- $\sigma \in (\Sigma^1 \setminus \Sigma^2) \cup \{\tau\}$ and $(q_1, \sigma, q'_1) \in \rightarrow_1$ and $q_2 = q'_2$
- $\sigma \in (\Sigma^2 \setminus \Sigma^1) \cup \{\tau\}$ and $(q_2, \sigma, q'_2) \in \rightarrow_2$ and $q_1 = q'_1$

□

In this definition of composition, a shared label must be an input label of one system and an output label of the other (except the silent label τ). The two systems must synchronize on shared labels which are then considered as internal in the composition. This corresponds to *rendez-vous* communication. Other communication paradigms can be modeled, such as *broadcast* where the common labels are put into the output set of the composed system. The choice of a communication schema is orthogonal to the issues studied here and would not modify the presented results.

The composition operator $\|$ is commutative, up to a renaming of the states: the STTS $\mathcal{A}\|\mathcal{B}$ is equal to $\mathcal{B}\|\mathcal{A}$ if its states (q_a, q_b) are renamed (q_b, q_a). Similarly, composition is associative if common labels are shared by at most two STTS. Let $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ be three STTS without common label (that is $\Sigma_\mathcal{A} \cap \Sigma_\mathcal{B} \cap \Sigma_\mathcal{C} = \{\tau\}$). If $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ are pairwise composable, then $\mathcal{A}\|(\mathcal{B}\|\mathcal{C})$ is equal to $(\mathcal{A}\|\mathcal{B})\|\mathcal{C}$ if its states $(q_a, (q_b, q_c))$ are renamed $((q_a, q_b), q_c)$.

An important property of composition is that it preserves the simulation relations. In combination with associativity, this property enables the modular design and verification of controllers. The definition of simulation is extended to STTS in the natural way. For weak simulation, the internal events are hidden (see Definition 2.8) and considered as being silent.

Definition 4.5 [(Weak) simulation] Let $\mathcal{S}_1$ be a STTS given by $\mathcal{T}_1$ and structured by $(\Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau)$, and $\mathcal{S}_2$ be a STTS given by $\mathcal{T}_2$ and structured by $(\Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau)$. We say that $\mathcal{S}_1$ (weakly) simulates $\mathcal{S}_2$, noted $\mathcal{S}_1 \succeq \mathcal{S}_2$ (resp. $\mathcal{S}_1 \succeq_{weak} \mathcal{S}_2$), if $\Sigma^1_{\mathsf{in}} = \Sigma^2_{\mathsf{in}}$ and $\Sigma^1_{\mathsf{out}} = \Sigma^2_{\mathsf{out}}$ and if $\mathcal{T}_1 \succeq \mathcal{T}_2$ (resp. $\mathcal{T}_1[\Sigma^1_\tau := \tau] \succeq_{weak} \mathcal{T}_2[\Sigma^2_\tau := \tau]$). □

Theorem 4.6 *Let $(\mathcal{S}_1, \mathcal{S}_2)$ and $(\mathcal{S}'_1, \mathcal{S}'_2)$ be two pairs of composable* STTS *such that $\mathcal{S}'_1 \succeq \mathcal{S}_1$ and $\mathcal{S}'_2 \succeq \mathcal{S}_2$. Then $\mathcal{S}'_1\|\mathcal{S}'_2 \succeq \mathcal{S}_1\|\mathcal{S}_2$.*

Proof (Sketch). Let R_1 and R_2 be simulation relations for $\mathcal{S}'_1 \succeq \mathcal{S}_1$ and $\mathcal{S}'_2 \succeq \mathcal{S}_2$ respectively. It is easy to show that $R_{12} = \{((q'_1, q'_2), (q_1, q_2)) \mid (q'_1, q_1) \in R_1 \wedge (q'_2, q_2) \in R_2\}$ is a simulation relation for $\mathcal{S}'_1 \| \mathcal{S}'_2 \succeq \mathcal{S}_1 \| \mathcal{S}_2$. ■

Notice that the properties of input enabledness and non-zenoness are not necessarily preserved by (weak) simulations. This is not surprising as we have seen that those properties are related to existential liveness (there must *exist* some non-blocking trajectory) while simulations preserve safety (*all* the trajectories avoid the bad states).

On the other hand, input enabledness is preserved by composition, and moreover input enabledness of two STTS ensures that their composition is non-blocking. A proof of Theorem 4.7 is given in Appendix A.1.

Theorem 4.7 *Let $\mathcal{S}_1, \mathcal{S}_2$ be two input-enabled and composable* STTS *(structured by $(\Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau)$ and $(\Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau)$ respectively). Let $\rightarrow^1$, $\rightarrow^2$ and $\rightarrow$ be the transition relations of $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_1 \| \mathcal{S}_2$ respectively. If $(q_1, q_2) \in \mathsf{Reach}(\mathcal{S}_1 \| \mathcal{S}_2)$ and there is a discrete transition $(q_1, \sigma, q'_1) \in \rightarrow^1$ with $\sigma \notin \Sigma^1_{\mathsf{in}}$ (respectively $(q_2, \sigma, q'_2) \in \rightarrow^2$ with $\sigma \notin \Sigma^2_{\mathsf{in}}$) then there exists a state q'_2 (resp. a state q'_1) such that $((q_1, q_2), \sigma, (q'_1, q'_2)) \in \rightarrow$.*

However, it is not true in general that the composition of two STTS that are input enabled and non-zeno is itself non-zeno. Nevertheless, that property holds for the particular STTS that we consider in the sequel (namely in Section 4.3.1).

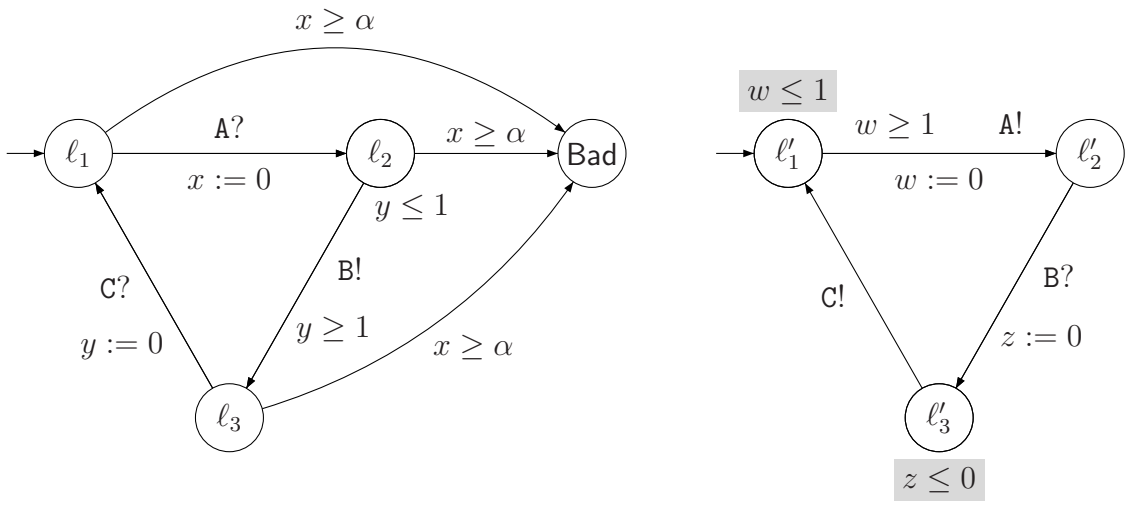
4.2.2 Structured Timed automata

We refine the definition of timed automata to deal with *open* controllers, whose alphabet is structured in sets of input, output and internal labels.

Definition 4.8 A timed automaton A with alphabet Lab is *structured* by a triple $(\mathsf{Lab}_{\mathsf{in}}, \mathsf{Lab}_{\mathsf{out}}, \mathsf{Lab}_\tau)$ if those three sets form a partition of Lab (with $\tau \in \mathsf{Lab}_\tau$). Its semantics is the STTS given by $[\![A]\!]$ and structured by $(\mathsf{Lab}_{\mathsf{in}}, \mathsf{Lab}_{\mathsf{out}}, \mathsf{Lab}_\tau)$. □

Running example We illustrate this chapter with a running example, shown in Figure 4.1. We have a model of an environment given in Figure 4.1(a), in the form of a structured timed automaton. We use the notation $\sigma!$ to denote *actions* that are triggered by the automaton ($\sigma \in \mathsf{Lab}_{\mathsf{out}}$) and $\sigma?$ to denote *events* that are received ($\sigma \in \mathsf{Lab}_{\mathsf{in}}$). The environment is supposed to be input enabled, but we do not depict the edges that are going to the Bad location, with a label $\sigma \in \mathsf{Lab}_{\mathsf{in}}$. The location Bad is the final state of the automaton, thus the state to avoid.

The environment starts in location ℓ_1 with $x = y = 0$ and waits for the event A. In location ℓ_2, the response B is sent when $y = 1$, and then the event C is accepted which resets the clock y and the automaton gets back to ℓ_1. At any moment in the



(a) The environment.

(b) The ASAP controller.

Figure 4.1: Running example.

cycle, there is an edge labelled by τ that leads to Bad with a guard $x \geq \alpha$ where α is a parameter. In the sequel, we study the correctness of the controller of Figure 4.1(b) when $\alpha \in \mathbb{Q}^{\geq 0}$.

To be correct, the controller must produce an event A at least every α time units, and after receiving the event B, has to output a C. That sequence of events must be done fast enough to avoid the clock x of the environment to run over α. One solution is given in Figure 4.1(b). The designer has chosen here to output an A every 1 time unit, and to react to the event B as quickly as possible.

Given this controller for the system, our task is to verify that it is correct and implementable. We define correctness and implementability later in this chapter, but intuitively the correctness asks that the environment avoids entering the location Bad and the implementability asks in addition that some amount of time is allowed to the controller to fire a transition and observe events.

4.3 Real-Time Implementations

In the classical semantics of timed automata (Definition 2.12), the invariants are used to bound the time the automaton can stay in a location, thus forcing to take a discrete transition. So, invariants are used to *ensure progress*. However, the progress is naturally present in an implementation, where the program is going to systematically take any transition as soon as it is enabled (which makes sense only for left

closed predicates). Therefore, the invariants in timed automata are not necessary if we make a *progress assumption* saying that transitions are executed as soon as possible. This assumption should be encoded in the semantics of structured timed automata. It should be noticed that the progress assumption should allow some delay to execute a transition, because this corresponds to realistic implementations. We emphasize the absence of invariants for timed controllers in the definition below. We also assume that the bad states are given in the specification of the environment, and that there is at most one initial location in the controller.

Definition 4.9 [ELASTIC controller] An ELASTIC controller is a structured Alur-Dill automaton $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ such that $\mathsf{Inv}(\ell) = \top$, $\mathsf{Final}(\ell) = \bot$ for every $\ell \in \mathsf{Loc}$ and $\mathsf{Init}(\ell)$ is satisfiable for at most one location $\ell \in \mathsf{Loc}$. □

Implementability of real-time models is not a well defined notion in the literature, as it may depend on various arbitrary choices, such as the type of hardware, the scheduling policy, interrupts handling, sensors polling, time management, etc. Therefore, the notion of implementability is in essence controversial, and a widely accepted definition is difficult to find. We first give a short overview of the important points that characterize "real" implementations.

The definition must be sufficiently close to the real implementations so that practitioners agree that an implementable system is indeed realizable. On the other hand, it must be sufficiently abstract to be independent of small internal differences between real-time platforms.

In this context, we propose a definition that is as simple as possible, and hopefully sufficiently general so that almost every implementation can be simulated by it. Hence, we do not focus on optimization issues.

We give an overview of the classical way real-time programs are implemented, influenced by our experience with a simple real-time system, the LEGO MINDSTORMS™ robots, and other real-time frameworks [Die99, Lab02].

The fundamental mechanism underlying those systems is *polling*. The operating system must periodically check the sensors to detect an input and activate the threads ready to execute. The scheduling policy is a round Robin, each thread receiving the CPU successively. The use of priorities somewhat complicates this picture, but the principle is the same.

A program consists of a set of *threads*, that is sequential programs executed in parallel (but not simultaneously on single-processor architectures) and typically sharing memory. For implementing ELASTIC controllers, we only use very simple forms of threads. Indeed, for a timed automaton, a thread would be the code corresponding to firing one transition, and the common data be the current location of the automaton and the value of the clocks. We give more details below on how we implement timed automata.

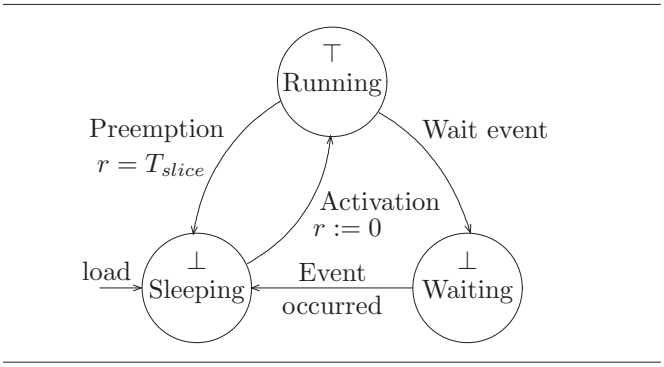


Figure 4.2: The three states of a thread.

The round Robin is implemented by time-slicing, that is assigning the CPU to one thread execution for a small amount of time called the *time-slice*, after which the thread is interrupted and put in *sleeping* mode, and the CPU is assigned another thread. In this work, we assume that the execution of a transition is always less than the time-slice. This is clearly realistic for simple examples, because the code for a transition is fairly simple (there is no alternative, nor complex constructions like function calls). But still, this should be justified in practice by an analysis of the sequential code generated by the compiler, namely by bounding the execution time of each instruction. An interesting future work would be to deal with execution times greater than the time-slice, which corresponds to consider an implicit notion of *task*.

As we mentioned, a reactive program is not intended to terminate. Therefore, after executing a transition, the thread is not deleted: rather it is *waiting* that the conditions are met to execute the transition again. Those conditions include being in the source location of the transition, and satisfying its guard. A real-time hardware has a single *system clock*, say $\mathtt{T}$, whereas timed automata may have several clocks. Therefore, we use the system clock as the absolute time, and we use the difference between the current time and the resetting time of a clock to obtain its current value. Thus, guards are evaluated using the value $\mathtt{T} - r(x)$ for each clock x where $r(x)$ is the last resetting time of x. Since the system clock is digital (with a precision of 1ms in the case of LEGO MINDSTORMS™), the number $\mathtt{T} - r(x)$ is an integer multiple of the unit of time.

In Figure 4.2, we give an informal picture of this schema. Initially, threads are loaded in *sleeping* state and successively receive the CPU, going to *running* mode. When the time a thread is running reaches the time-slice, the thread is interrupted, getting back to sleeping mode (this does not occur for our timed automata). If a thread has "terminated" before the time-slice and is thus waiting for an event, it goes to the state *waiting*. The operating system then checks the new inputs and moves the corresponding waiting threads to sleeping. Then, the next sleeping thread is run.

In the case of LEGO MINDSTORMS™, we use semaphores to guarantee mutual exclusive access to shared data, and in particular, a boolean semaphore is associated to each location of the controller. One of those semaphores is set to 1 corresponding to the current location. Every transition is waiting that the semaphore of its source location equals 1. If a transition is enabled, its thread goes to sleeping and the location semaphore is decremented. This is done with an atomic *test and set* procedure (similar to the POSIX sem_wait) so that at most one thread can be put in sleeping mode at each round. We also use boolean semaphores to encode events. In Figure 4.3, we give a high-level abstract model of the thread coding a transition $(\ell, \ell', g, \sigma, R)$, where either $\sigma \in \mathsf{Lab_{in}}$ is an input event or $\sigma \in \mathsf{Lab_{out}}$ is an output event. In this code, the function *wait_event* is used to *test* both the semaphore of the location ℓ (and the semaphore of σ if $\sigma \in \mathsf{Lab_{in}}$), and the condition expressed by the guard g. If the two (or three) conditions are satisfied, the semaphore(s) is (are) *atomically set* to *false*. The rest of the while loop executes the transition, resetting the clocks in R (assigning them the value of the system clock), posting the semaphore of ℓ' (that is, setting it to *true*) and optionally posting the semaphore of σ (if $\sigma \in \mathsf{Lab_{out}}$).

In the description we have given of the implementation of timed automata under LEGO MINDSTORMS™, many aspects would be shared by other platforms, but probably slightly different solutions are also satisfying. Since we want to be as much as possible independent of a particular platform, we take a definition of implementability that does not account for the very low details of the internals of the target hardware and operating system. First, we define a *program semantics* for structured timed automata.

```
while True do
    wait_event(ℓ, σ, g);
    foreach x ∈ R do x ← 0;
    post(ℓ');
```

(a) if $\sigma \in \mathsf{Lab_{in}}$.

```
while True do
    wait_event(ℓ, g);
    foreach x ∈ R do x ← 0;
    post(σ);
    post(ℓ');
```

(b) if $\sigma \in \mathsf{Lab_{out}}$.

Figure 4.3: Thread for the transition $(\ell, \ell', g, \sigma, R)$.

4.3.1 Program semantics

The program semantics is a formal semantics for the following interpretation procedure of structured timed automata. This procedure repeatedly executes what we call *execution rounds*. An execution round is defined as follows:

1. the current system time is read in the clock register of the CPU and stored in the variable `T`;
2. the input sensors are checked for new events issued by the environment and the list of input events to handle is updated;
3. the guards of the transitions leaving the current location are evaluated with the value stored in `T`. If at least one guard evaluates to true then one of the enabled transitions is chosen nondeterministically and executed;
4. the next round is started.

We choose this semantics for its simplicity and because it is obviously implementable. All we require from the hardware is to respect the following two requirements: (i) the clock register of the CPU is incremented every Δ_P time units and (ii) the time spent in one loop is bounded by a fixed value Δ_L . The values $\Delta_P = 1$ms and $\Delta_L = 6$ms are typical for LEGO MINDSTORMS™.

Our program semantics is close to the semantics of PLC-automata [Die01]. We give a detailed comparison with that framework in Section 4.5.

As we have mentioned, our model of time is continuous, but the implementations have only access to discrete time through the digital clock producing *ticks* every Δ_P. Note however that the environment can issue events at any point between two ticks. Thus, the semantics must be continuous time, even though the program executes in a discrete time fashion, and with variables that have a fixed precision. Thus, the program has a *sampled* view of the environment. This notion is captured by the following notations.

Definition 4.10 [Clock rounding] For $T \in \mathbb{R}^{\geq 0}$ and $\delta \in \mathbb{R}^{>0}$, let $\lfloor T \rfloor$ (resp. $\lceil T \rceil$) be the greatest (resp. lowest) integer k such that $k \leq T$ (resp. $k \geq T$) and let $\lfloor T \rfloor_\delta = \delta \lfloor \frac{T}{\delta} \rfloor$ and $\lceil T \rceil_\delta = \delta \lceil \frac{T}{\delta} \rceil$. □

In the implementation, guards of transitions are evaluated periodically. Thus, if a guard does not remain true for a sufficiently long time, it is likely that the implementation will miss it. Therefore, we *enlarge* the guards to guarantee that a satisfied guard is eventually detected by the polling loop of the program.

Definition 4.11 [Predicate transformations] Let $\delta_1, \delta_2 \in \mathbb{Q}$. For a rectangular predicate $\varphi \in \mathsf{Rect}(\mathsf{Var})$, we use the symbols $\langle \in \{[,]\}$ and $\rangle \in \{],[\}$ to define the notation

$$ {}_{\delta_1}\langle \varphi \rangle_{\delta_2} $$

as follows (recursively):

- ${}_{\delta_1}\langle \top \rangle_{\delta_2} \equiv \top$;

- ${}_{\delta_1}\langle\bot\rangle_{\delta_2} \equiv \bot$;
- ${}_{\delta_1}\langle x \geq a\rangle_{\delta_2} \equiv {}_{\delta_1}\langle x > a\rangle_{\delta_2} \equiv \begin{cases} x \geq a - \delta_1 & \text{if } \langle = [\\ x > a - \delta_1 & \text{if } \langle =] \end{cases}$
- ${}_{\delta_1}\langle x \leq a\rangle_{\delta_2} \equiv {}_{\delta_1}\langle x < a\rangle_{\delta_2} \equiv \begin{cases} x \leq a + \delta_2 & \text{if } \rangle =] \\ x < a + \delta_2 & \text{if } \rangle = [\end{cases}$
- ${}_{\delta_1}\langle\varphi_1 \wedge \varphi_2\rangle_{\delta_2} \equiv {}_{\delta_1}\langle\varphi_1\rangle_{\delta_2} \wedge {}_{\delta_1}\langle\varphi_2\rangle_{\delta_2}$

□

For example, for $\delta_1 = -\frac{1}{3}$ and $\delta_2 = \frac{1}{2}$, we have:

$$ {}_{\delta_1}]2 \leq x \leq 5]_{\delta_2} \equiv 2 + \frac{1}{3} < x \leq 5 + \frac{1}{2} \equiv \frac{7}{3} < x \leq \frac{11}{2} $$

We are now ready to define the program semantics. The state space of the program semantics contains two types of informations: (*i*) the state of the automaton: the current location ℓ, the resetting time $r(x)$ of each clock x, the time $I(\sigma)$ the input event σ has been pending for ($I(\sigma) = \bot$ if σ is not pending) and the time d elapsed since the last location change; (*ii*) the state of the program: the absolute time T at the beginning of the current execution round, the time u spent in the current round, and a boolean f saying whether a transition has occurred in the current round.

All the timing informations contained in (r, I, d, T, u) are real numbers, and represent *exact* values. Since the program manipulates discrete variables, when we use those values in the implementation, we have to round the exact timing informations to the precision Δ_P of the system clock, for example to evaluate a guard.

We give more intuitions right after the definition.

Definition 4.12 [Program semantics] Let $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final}\rangle$ be an ELASTIC controller structured by $(\mathsf{Lab}_{\mathsf{in}}, \mathsf{Lab}_{\mathsf{out}}, \mathsf{Lab}_{\tau})$, and let Δ_L, $\Delta_P \in \mathbb{Q}^{>0}$. Let $\Delta_S = \Delta_L + \Delta_P$. The program semantics of A is the STTS $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P}$ given by $\langle Q, Q_0, Q_f, \Sigma, \rightarrow\rangle$ structured by $(\Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_\tau)$, where:

- $Q = \{(\ell, r, I, d, T, u, f) \mid \ell \in \mathsf{Loc} \wedge r : \mathsf{Var} \to \mathbb{R}^{\geq 0} \wedge I : \mathsf{Lab}_{\mathsf{in}} \to \mathbb{R}^{\geq 0} \cup \{\bot\} \wedge d, T, u \in \mathbb{R}^{\geq 0} \wedge f \in \{\top, \bot\}\}$;
- $Q_0 = \{(\ell, r, I_0, 0, 0, 0, \bot) \mid -r \in \mathsf{Init}(\ell)$ and $I_0(\sigma) = \bot$ for every $\sigma \in \mathsf{Lab}_{\mathsf{in}}\}$;
- $Q_f = \varnothing$;
- $\Sigma_{\mathsf{in}} = \mathsf{Lab}_{\mathsf{in}}$, $\Sigma_{\mathsf{out}} = \mathsf{Lab}_{\mathsf{out}}$, and $\Sigma_\tau = \mathsf{Lab}_\tau \cup \overline{\mathsf{Lab}_{\mathsf{in}}} \cup \{\tau\}$;

- the transition relation is defined as follows: for $q = (\ell, r, I, d, T, u, f)$, we have $(q, \sigma, q') \in \rightarrow$ if and only if

  – for the discrete transitions,

  ($P1$) either $\sigma \in \mathsf{Lab_{out}} \cup \mathsf{Lab}_\tau$ and $f = \bot$, $q' = (\ell', r', I, 0, T, u, \top)$ and there exists $(\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$ such that $\lfloor T \rfloor_{\Delta_P} - \lfloor r \rfloor_{\Delta_P} \models {}_{\Delta_S}[g]_{\Delta_S}$ and $r' = r[R := T]$;

  ($P2$) or $\sigma \in \mathsf{Lab_{in}}$ and $q' = (\ell, r, I', d, T, u, f)$ where $I' = I$ if $I(\sigma) \neq \bot$ and $I' = I[\sigma := 0]$ if $I(\sigma) = \bot$;

  ($P3$) or $\sigma = \bar{\alpha} \in \overline{\mathsf{Lab_{in}}}$ and $I(\alpha) > u$, $f = \bot$, $q' = (\ell', r', I', 0, T, u, \top)$ with $I' = I[\sigma := \bot]$ and there exists $(\ell, \ell', g, \alpha, R) \in \mathsf{Edg}$ such that $\lfloor T \rfloor_{\Delta_P} - \lfloor r \rfloor_{\Delta_P} \models {}_{\Delta_S}[g]_{\Delta_S}$, $r' = r[R := T]$;

  ($P4$) or $\sigma = \tau$ and $q' = (\ell, r, I, d, T + u, 0, \bot)$, and either $f = \top$ or the two following conditions hold:

  ($P4.1$) for all $\alpha \in \mathsf{Lab_{in}}$, for all $(\ell, \ell', g, \alpha, R) \in \mathsf{Edg}$, we have that either $\lfloor T \rfloor_{\Delta_P} - \lfloor r \rfloor_{\Delta_P} \not\models {}_{\Delta_S}[g]_{\Delta_S}$ or $I(\alpha) \leq u$;

  ($P4.2$) for all $\sigma \in \mathsf{Lab_{out}} \cup \mathsf{Lab}_\tau$, for all $(\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$, we have that $\lfloor T \rfloor_{\Delta_P} - \lfloor r \rfloor_{\Delta_P} \not\models {}_{\Delta_S}[g]_{\Delta_S}$

  – for the timed transitions,

  ($P5$) $\sigma = t \in \mathbb{R}^{\geq 0}$ and $q' = (\ell, r, I + t, d + t, T, u + t, f)$ and $u + t \leq \Delta_L$.

□

Comments on the program semantics. First, since the program is started with $T = 0$, the initial value of a clock x in the program is $T - r(x) = -r(x)$. Therefore, we ask in the definition of the initial states Q_0 that $-r \in \mathsf{Init}(\ell)$.

We observe that guards are enlarged by $\Delta_S = \Delta_L + \Delta_P$. This increases the enabling time of the guard and so it ensures that the guard is not missed by the program.

The guards are evaluated with the valuation $v_1 = \lfloor T \rfloor_{\Delta_P} - \lfloor r \rfloor_{\Delta_P}$ which gives, up to Δ_P, the time elapsed since the last reset of each variable. If the guard is satisfied in the program at the beginning of a round, the transition could be taken until the end of the round, thus at most Δ_L time units later. At that time, the exact valuation of the clock (which is not visible to the program) is $v_2 = T + u - r$. The distance $|v_2 - v_1|$ is thus bounded by $\Delta_S = \Delta_L + \Delta_P$.

Rule ($P1$) asserts that an output or internal action can be issued provided such a labeled transition is enabled in the current state of the program, and we have not yet taken a transition ($f = \bot$). Clocks are reset to the stored absolute time T.

Rule ($P2$) ensures input enabledness. The occurrence of an event from the environment is stored in I, provided no such event is already pending, that is $I(\sigma) = \bot$.

Otherwise, the event is accepted but I is not modified. This way we define a queue I with bounded size and, to make the presentation simpler, bounded by 1. If the size of the queue was unbounded, we fall into a class of systems that are infinite in their discrete states, and studying such systems is out of the scope of this thesis.

In Rule $(P3)$, the label $\bar{\alpha}$ means that the controller has handled the input event α and thus has executed a transition labelled by α. We say that an event α is visible to the program if it has occurred *before* the beginning of the current round (that is $I(\alpha) > u$, assuming that $a > \bot$ for all $a \in \mathbb{R}$). The event is then removed from the queue ($I'(\alpha) = \bot$). The rest of the rule is similar to $(P1)$. We always assume that $\overline{\mathsf{Lab}_{\mathsf{in}}}$ and Lab are disjoint.

In Rule $(P4)$, the silent label τ is used to terminate the current execution round. We update the variable T and the flag f, at the condition that either we did take a transition ($f = \top$), or there was no enabled transition at all, neither labelled by an input event $(P4.1)$, nor by an output or internal action $(P4.2)$. This rule forces the program to progress when possible.

The timed transitions are allowed by Rule $(P5)$. The time spent in a round is bounded by Δ_L and thus, a new round must be started to let more time elapse. Notice that this is always possible by Rule $(P4)$. Finally, note that in every state the absolute time is given by $T + u$.

In this definition, the variable d recording the time spent in the current location is not necessary. In fact d is an additional information that we maintain along the execution, but that does not influence the program. More precisely, if $((\ell, r, I, d_1, T, u, f), \sigma, (\ell', r', I', d', T', u', f')) \in \rightarrow$ then for all $d_2 \in \mathbb{R}^{\geq 0}$, we have $((\ell, r, I, d_2, T, u, f), \sigma, (\ell', r', I', d', T', u', f')) \in \rightarrow$. The reason to introduce this variable will be clear in the next section.

Our definition of implementability relies on the program semantics. Remind that only the controller is implemented.

Definition 4.13 [Implementability of ELASTIC controllers] An ELASTIC controller A embedded in an STTS environment Env is *implementable* if there exists $\Delta_L, \Delta_P \in \mathbb{Q}^{>0}$ such that $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P} \| \mathsf{Env}$ is empty. □

Observe that the implementability is not a property of the controller alone: it also depends on the property to verify (which is hidden inside the specification of the environment, in the form of bad states). Moreover, it is easy to see that every ELASTIC controller A is non-zeno and we can show that the composition $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P} \| \mathsf{Env}$ for $\Delta_L, \Delta_P > 0$ is non-zeno as soon as Env is non-zeno.

Verifying the implementability of a timed controller using directly the program semantics suffers from two main drawbacks. First, the constants appearing in the semantics are likely to be of completely different scales. On the one hand, the hardware has very small execution times (*e.g.* $\Delta_L = 6$ms), and on the other hand, the timing

of the controllers we implement can typically be expressed in terms of seconds. This makes the analysis very difficult because the state space is unnecessarily *fragmented* by the model-checkers, see for example [IKL+00]. In the next section, we propose a general methodology that allows to verify the implementability, independently of the constants of the hardware, and without the fragmentation of the state space due to the repetition of a huge number of very small time steps. Second, it would be much more attractive if the constants Δ_L and Δ_P could be left as parameters during the verification. This way, we would *synthesize* a set of values for Δ_L and Δ_P that guarantee the correctness and we could *choose a posteriori* the characteristics of the hardware in order to satisfy the constraint. The problem with the program semantics is that it is not monotonic in its parameters, in the sense that it is not obvious that choosing smaller parameters preserves the correctness. In particular, it is not true that $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P} \succeq [\![A]\!]^{\mathsf{Prg}}_{\Delta'_L,\Delta'_P}$ if $\Delta_L > \Delta'_L$ and $\Delta_P > \Delta'_P$, which is however a desirable property. To see this, consider a controller with two locations ℓ_1,ℓ_2 and one edge $(\ell_1, \ell_2, \top, \alpha, \varnothing)$ where α is an input label. Assume that an event α is issued by the environment 11ms after the system is started (in location ℓ_1 for the controller). In a first implementation, let $\Delta'_L = 10$ms (the value of Δ_P does not matter here), and thus the transition is enabled in the third execution round. Assume that it is executed at time 27ms. In a second implementation with $\Delta_L = 12$ms, the transition is enabled in the second execution round and thus it must be executed between 12ms and 24ms. Hence it is impossible to simulate the faster implementation. A similar phenomenon can happen with output labels, for example if an enlarged guard ${}_{\Delta_S}[g]_{\Delta_S}$ is satisfied for the first time after 11ms.

In the methodology we develop further, this intuitive property that *faster is better* holds, and thus we have the guarantee that using a faster hardware is not a source of new errors. Also, we show in Chapter 6 that a parametric verification problem for timed automata that is tightly related to the question of implementability is decidable.

4.4 A New Semantics for Timed Automata

Running example As already pointed out in the introduction, the synchrony hypothesis is problematic if we want to transfer the properties verified on the model to an implementation. For timed automata, we illustrate the properties of the classical semantics (see Definition 2.12) that makes difficult to implement a controller while preserving the properties that are verified in the design.

First, note that the invariants of the controller (shaded on Figure 4.1(b)) are used to force the controller to take actions. The invariants can be removed if we make a *maximal progress* assumption for the controller: any action is taken as soon as it is enabled. So, for example, the transition labeled by `A!` is fired exactly when $z = 0$, *i.e.* instantaneously. Clearly, no hardware can avoid reaction delays. Second, synchronizations between the environment and the controller (*e.g.* transitions labeled by `B`) are not instantaneous as well. Some time is necessary to the hardware to detect

an incoming event B and to the software to take this event into account (through interruptions for example). Third, in continuous time we assume that the clocks of the controller are infinitely precise (their value ranges over a dense domain), which is not realistic. It is not obvious that a digital clock can replace a real-valued clock without changing the reachability properties of the system. Finally, the classical semantics is not input enabled, as it does not take into account the structuration of timed automata.

Those problems are similar to the synchrony hypothesis and show that even if we have formally verified our control strategy using the classical semantics for timed automata (and thus under the synchrony hypothesis), we cannot conclude that an implementation will preserve the properties that we have proven on the model. This is very unfortunate. To summarize, there are two options offered to the designer:

- either he gives up the synchrony hypothesis and he models the target real-time platform that implements the controller, as for example in [IKL+00],
- or he goes on with the synchrony hypothesis at the modeling level, but during the verification phase, the ASAP semantics is relaxed in order to *formally validate the synchrony hypothesis*.

We think that the second option is much more appealing and we propose a framework that makes it possible *theoretically*, but also feasible *practically*. The framework that we propose is a relaxation of the ASAP semantics that we call the Almost ASAP semantics (or AASAP semantics).

We formally define this semantics and we show that:

- it is *robust* in the sense that it defines a *tube of strategies* (instead of a unique strategy as in the ASAP semantics) which can be refined into an implementation that preserves the safety properties.
- it can be encoded in classical timed automata, and thus verified in practice with the existing model-checkers.

4.4.1 Requirements

At the modeling level, when writing down the system specification, it is very important to concentrate on the conceptual aspects: the discrete structure (the locations), the set of events that are awaited, the actions to be issued, the timing constraints, the concurrency, etc. At that point, it is often very convenient to forget about some low level details, therefore assuming for example that the clocks are perfectly precise, that executing an assignment takes no time, that time is continuous, etc.

As an illustration, consider the strict inequalities in timing constraints. It is simpler to write $x > 1$ where we mean $x \geq 1 + \epsilon$ with $\epsilon > 0$, even though we slightly over-approximate the constraint in mind. In an implementation however, it is not clear

what is the difference between $x > 1$ and $x \geq 1$ when x is a (continuous) clock. Thus, strict inequalities are very useful for the design of the system, but have no realistic counterpart in implementations.

As we mentioned in the previous section, implementations are executing with a kind of almost ASAP strategy that ensures a progress assumption. Enabled transitions are not executed urgently whenever they are enabled, but urgently within some fixed delay δ after being enabled.

To formally define that almost urgency, we need to know the time a transition has been enabled for. If this amount of time is greater than δ then the transition is urgent. Three elements are necessary to determine whether an enabled transition is urgent in the current state: (*i*) the time we have been in the current location (*ii*) the current valuation of the clocks, and optionally (*iii*) the release time of the input events issued by the environment (in the case the transition synchronizes on such an event).

Therefore, in addition to the usual location ℓ and valuation v, we have to remember the time $d \in \mathbb{R}^{\geq 0}$ spent in the current location, and the time elapsed after the occurrence of each event (for those events that occurred), in the form of a function $I : \mathsf{Lab_{in}} \to \mathbb{R}^{\geq 0} \cup \{\bot\}$, the special value $\bot$ being returned for non pending events. In a state (ℓ, v, I, d), the enabled time of a transition $(\ell, \ell', g, \sigma, R)$, where $\sigma \in \mathsf{Lab_{in}}$ is an input event, is the minimum of the three values d, $I(\sigma)$ ($\bot$ is less than any number) and $\mathsf{ETime}(g, v)$ (where $\mathsf{ETime}(g, v)$ denotes the time the guard g has been *enabled for* in a state with current valuation v).

Definition 4.14 Given a valuation $v : \mathsf{Var} \to \mathbb{R}^{\geq 0}$ and a closed rectangular predicate $\varphi \in \mathsf{Rect_c}(\mathsf{Var})$, let

$$\mathsf{ETime}(\varphi, v) = \sup\{t \in \mathbb{R}^{\geq 0} \mid \forall t' \in [0, t] : \ v - t' \models \varphi\}$$

(with the usual convention that $\sup \varnothing = -\infty$). □

This definition is illustrated on Figure 5.7.

4.4.2 Almost ASAP semantics

According to the previous discussions, we can summarize the main characteristics of the new semantics as follows. The Almost ASAP semantics is:

- input-enabled, accepting any input in every state;
- non-zeno and, moreover, some delay δ can always pass after a discrete transition;
- urgent within δ time units, in the sense that a transition is urgent when it has been enabled for a time at least δ.

We give detailed comments on the AASAP semantics after the formal definition.

Definition 4.15 [AASAP semantics] Let $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ be an ELASTIC controller structured by $(\mathsf{Lab}_{\mathsf{in}}, \mathsf{Lab}_{\mathsf{out}}, \mathsf{Lab}_{\tau})$, and let $\delta \in \mathbb{Q}^{\geq 0}$. The AASAP *semantics* of A is the STTS $[\![A]\!]_{\delta}^{\mathsf{AAsap}}$ given by $\langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$ structured by $(\Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_{\tau})$ where:

- $Q = \{(\ell, v, I, d) \mid \ell \in \mathsf{Loc} \wedge v : \mathsf{Var} \rightarrow \mathbb{R}^{\geq 0} \wedge I : \Sigma_{\mathsf{in}} \rightarrow \mathbb{R}^{\geq 0} \cup \{\bot\} \wedge d \in \mathbb{R}^{\geq 0}\}$;
- $Q_0 = \{(\ell, v, I_{\bot}, 0) \mid \ell \in \mathsf{Loc} \wedge v \in [\![\mathsf{Init}(\ell)]\!] \wedge \forall \sigma \in \Sigma_{\mathsf{in}} : I_{\bot}(\sigma) = \bot\}$;
- $Q_f = \varnothing$;
- $\Sigma_{\mathsf{in}} = \mathsf{Lab}_{\mathsf{in}}$, $\Sigma_{\mathsf{out}} = \mathsf{Lab}_{\mathsf{out}}$, and $\Sigma_{\tau} = \mathsf{Lab}_{\tau} \cup \overline{\mathsf{Lab}_{\mathsf{in}}} \cup \{\tau\}$;
- The transition relation is defined as follows:
  - for the discrete transitions, we have $((\ell, v, I, d), \sigma, (\ell', v', I', d')) \in \rightarrow$ iff
    - $(A1)$ either $\sigma \in \mathsf{Lab}_{\mathsf{out}} \cup \mathsf{Lab}_{\tau}$, $I' = I$, $d' = 0$ and there exists $(\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$ such that $v \models_{\delta} [g]_{\delta}$ and $v' = v[R := 0]$;
    - $(A2)$ or $\sigma \in \mathsf{Lab}_{\mathsf{in}}$, $(\ell', v', d') = (\ell, v, d)$, and either $I(\sigma) \neq \bot$ and $I' = I$ or $I(\sigma) = \bot$ and $I' = I[\sigma := 0]$;
    - $(A3)$ or $\sigma = \bar{\alpha} \in \overline{\mathsf{Lab}_{\mathsf{in}}}$, $d' = 0$, $I(\alpha) \neq \bot$, $I' = I[\alpha := \bot]$ and there exists $(\ell, \ell', g, \alpha, R) \in \mathsf{Edg}$ such that $v \models_{\delta} [g]_{\delta}$ and $v' = v[R := 0]$;
    - $(A4)$ or $\sigma = \tau$ and $(\ell', v', I', d') = (\ell, v, I, d)$.
  - for the timed transitions, we have $((\ell, v, I, d), t, (\ell, v', I', d')) \in \rightarrow$ for $t \in \mathbb{R}^{\geq 0}$ iff $v' = v + t$, $I' = I + t$ (where $\bot + t = \bot$), $d' = d + t$ and
    - $(A5.1)$ for all edges $(\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}_{\mathsf{out}} \cup \mathsf{Lab}_{\tau}$,we have:

$$\forall t' \in [0, t] : d + t' \leq \delta \vee \mathsf{ETime}(g, v + t') \leq \delta$$

    - $(A5.2)$ and for all edges $(\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}_{\mathsf{in}}$, we have:

$$\forall t' \in [0, t] : d + t' \leq \delta \vee \mathsf{ETime}(g, v + t') \leq \delta \vee (I + t')(\sigma) \leq \delta$$

□

Comments on the AASAP semantics. Rule $(A1)$ corresponds to issuing an output action or an internal event, through a transition whose guard is enlarged by δ. Clocks are reset according to the transition taken, and the delay d for a location change is also set to 0.

Rules $(A2)$ and $(A3)$ are dealing with input events. The environment controls the occurrence of those events, and thus by Rule $(A2)$ the semantics must accept them at any moment (input enabledness). On the other hand, by Rule $(A3)$ an event α can be treated as soon as it is pending ($I(\alpha) \neq \bot$) and there is an enabled transition labelled by α in the controller. The event is then removed from the queue ($I'(\alpha) = \bot$) and the other variables are updated similarly to Rule $(A1)$. For technical reasons, we allow silent self-loops by Rule $(A4)$.

Rule $(A5)$ controls the passage of time. As soon as no transition is *urgent*, time can pass freely. A transition $(\ell, \ell', g, \sigma, R)$ with $\sigma \in \mathsf{Lab_{out}} \cup \mathsf{Lab}_\tau$ is urgent if it has been enabled for more than δ time units, that is (i) the current location was ℓ for at least δ time units ($d > \delta$) and (ii) its guard g has been satisfied by the current valuation v for at least δ ($\mathsf{ETime}(g, v) > \delta$). If $\sigma \in \mathsf{Lab_{in}}$, we ask in addition that (iii) the event σ has occurred δ time units ago ($I(\sigma) > \delta$). Rule $(A5.1)$ asks that all the transitions with $\sigma \in \mathsf{Lab_{out}} \cup \mathsf{Lab}_\tau$ are *not* urgent, and Rule $(A5.2)$ asks that all the transitions with $\sigma \in \mathsf{Lab_{in}}$ are *not* urgent.

In the next section, we give some important properties of the AASAP semantics. The key issue is that this semantics is an over-approximation of the program semantics. More precisely, the AASAP semantics $[\![A]\!]_\delta^{\mathsf{AAsap}}$ simulates the program semantics $[\![A]\!]_{\Delta_L,\Delta_P}^{\mathsf{Prg}}$, under some condition on the parameters Δ_L, Δ_P and δ. Therefore, verifying the correctness of a controller using the AASAP semantics gives for free a proof that the implementation of the controller is correct according to the program semantics.

Definition 4.16 [Robust correctness of ELASTIC controllers] For $\delta \in \mathbb{Q}^{\geq 0}$, an ELASTIC controller A embedded in an STTS environment Env is *correct up to* δ if $[\![A]\!]$ and Env are composable and $[\![A]\!]_\delta^{\mathsf{AAsap}} \| \mathsf{Env}$ is empty. □

Three problems can be formulated about the AASAP semantics: either (i) we know the characteristics of the hardware and so we *fix* the value of δ, or (ii) we ask whether there *exists* some δ such that the controller is correct, or (iii) we ask to *maximize* δ such that the controller is correct. In the context of verification, we assume that the environment is an Alur-Dill automaton (interpreted in the classical semantics) because the problems would already be undecidable if the environment was in the more general class of rectangular automata (see Definition 2.22).

Definition 4.17 [Robust safety control problems] Given a structured Alur-Dill automaton Env, and an ELASTIC controller A embedded in $[\![\mathsf{Env}]\!]$, the *robust safety control problem* asks:

- [**Fixed**] whether A is correct up to δ for a given fixed value of $\delta \in \mathbb{Q}^{\geq 0}$;
- [**Existence**] whether there exists $\delta \in \mathbb{Q}^{>0}$ such that A is correct up to δ;
- [**Maximization**] to maximize δ such that A is correct up to δ.

□

We show in Chapter 5 that the first flavour [Fixed] of the robust safety control problem is decidable, by reducing the problem to emptiness of Alur-Dill automata. The question of [Existence] is the subject of Chapter 6 where we show the decidability of a closely related question, by establishing a strong relation with a notion of robustness with regard to drifts in clocks. Note that we are interested in the existence of δ *strictly* positive. This will be justified by Corollary 4.21. We have no algorithmic solution to the question of [Maximization]. However, the maximal δ can be approximated up to any precision with a binary search based on the [Fixed] problem, if we assume that a bound on δ is known. Note that this is not sufficient to decide [Existence].

4.4.3 Properties of the AASAP semantics

We mention some important properties of the AASAP semantics. The detailed proofs can be found in [DDR04, DDR05a] and will appear in Martin De Wulf's PhD thesis.

We state a first property of the AASAP semantics corresponding to the informal statement *faster is better*. If a controller embedded in an environment is correct up to δ_1, then it is also correct up to δ_2 for any $\delta_2 \leq \delta_1$.

Theorem 4.18 *Let A be an* ELASTIC *controller, for all $\delta_1, \delta_2 \in \mathbb{Q}^{\geq 0}$ such that $\delta_2 \leq \delta_1$, we have $[\![A]\!]^{\mathsf{AAsap}}_{\delta_1} \succeq [\![A]\!]^{\mathsf{AAsap}}_{\delta_2}$.*

Theorem 4.18, Theorem 4.6 and Proposition 2.7 allow us to state the following corollary:

Corollary 4.19 *Let A be an* ELASTIC *controller embedded in an* STTS *environment* Env*. For all $\delta_1, \delta_2 \in \mathbb{Q}^{>0}$, if A is correct up to δ_1 and $\delta_2 \leq \delta_1$ then A is correct up to δ_2.*

Theorem 4.20 establishes that the program semantics is a refinement of the AASAP semantics. It gives a simple sufficient condition on the parameters Δ_L, Δ_P and δ for the AASAP semantics to simulate the program semantics.

Theorem 4.20 (Simulation) *Let A be an* Elastic *controller. For all $\delta, \Delta_L, \Delta_P \in \mathbb{Q}^{>0}$ such that $\delta \geq 3\Delta_L + 2\Delta_P$, we have $[\![A]\!]_\delta^{\mathsf{AAsap}} \succeq [\![A]\!]_{\Delta_L,\Delta_P}^{\mathsf{Prg}}$.*

The proof is quite involved and the bound $3\Delta_L + 2\Delta_P$ may not be tight as we have managed to use a simulation relation for $[\![A]\!]_\delta^{\mathsf{AAsap}} \succeq [\![A]\!]_{\Delta_L,\Delta_P}^{\mathsf{Prg}}$ as simple as possible. Nevertheless, it is clear that the tightest bound is at least $2\Delta_L + \Delta_P$ as a transition whose guard becomes true (up to Δ_P) just after the current round has been started could be executed only at the end of the next round.

Corollary 4.21 *Let A be an* Elastic *controller embedded in an* STTS *environment* Env. *If A is correct up to δ for some $\delta \in \mathbb{Q}^{>0}$ then A is implementable.*

In practice, either we want to use a given hardware with its characteristics (Δ_L, Δ_P) or we want to determine the slowest hardware able to implement the controller. In the first case, we use the [Fixed] flavour of the robust safety control problem with $\delta = 3\Delta_L + 2\Delta_P$, and in the second case we use the flavours [Existence] and [Maximization] to know if the controller is implementable at all, and compute the largest δ. In practice, verifying the AASAP semantics with a δ fixed may be difficult when δ is small with regard to the constants of the controller and of the environment. Problems like the fragmentation of the state space can be unmanageable for model-checkers. It is also preferable to have a procedure for maximization as it allows to choose the slowest hardware (or the cheapest) that is able to implement the controller

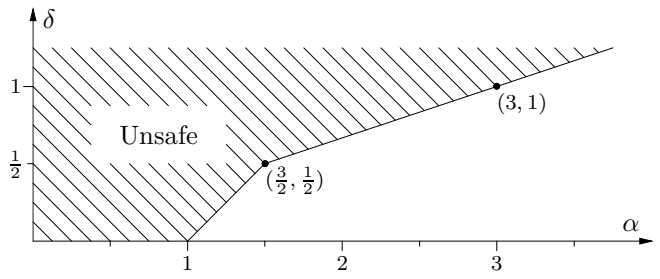


Figure 4.4: Parametric analysis of the running example (Figure 4.1).

Running example We illustrate Theorem 4.20 and Corollary 4.21 on the running example. Consider the diagram of Figure 4.4. It gives, as a function of the parameter α, the values of δ such that the AASAP semantics of the controller of Figure 4.1 is correct up to δ. It was obtained with HyTech, using the construction presented in Chapter 5. The condition can be expressed as $\delta \leq \min\{\alpha - 1, \frac{\alpha}{3}\}$. For the values of δ in the shaded region labelled "Unsafe", the system is not correct, that is the location Bad is reachable. In particular for $\alpha < 1$, the system is not correct for any δ. For $\alpha = 1$ and

$\delta = 0$, the controller is correct, but according to Corollary 4.21, it is not implementable. It means that the correct control strategies for $\alpha = 1$ are not reasonable, because they block time, or they are zeno, or they do not let a minimal amount of time between events. For example, consider the following sequence of transitions for a trajectory of the system that avoids Bad:

$$\underbrace{1, \mathtt{A}, \overbrace{0}^{1}, \mathtt{B}, \tfrac{1}{2}, \mathtt{C}}_{1}, \underbrace{\tfrac{1}{2}, \mathtt{A}, \overbrace{\tfrac{1}{2}}^{1}, \mathtt{B}, \tfrac{1}{4}, \mathtt{C}}_{1}, \underbrace{\tfrac{1}{4}, \mathtt{A}, \overbrace{\tfrac{3}{4}}^{1}, \mathtt{B}, \tfrac{1}{8}, \mathtt{C}}_{1}, \underbrace{\tfrac{1}{8}, \mathtt{A}, \tfrac{7}{8}}_{1}, \mathtt{B}, \tfrac{1}{16}, \mathtt{C}, \ldots t_i, \mathtt{A}, t'_i, \mathtt{B}, t''_i, \mathtt{C}, \ldots$$

where $t_i = \frac{1}{2^i}$, $t'_i = 1 - t_i$ and $t''_i = \frac{1}{2^{i+1}}$ for $i = 0, 1, \ldots$. This is a trace of the system because (i) the time between two consecutive A's is equal to 1, as required by the controller (through clock w); (ii) the time between a C and the next B is equal to 1, as required by the environment (through clock y). On the other hand, the value of x in location ℓ_1 of the environment is $t'_i + t''_i = 1 - \frac{1}{2^{i+1}} < 1$ and thus the Bad location is unreachable. However, this trace is clearly not implementable because the reaction time after an input B for issuing C and A is not bounded from below. The condition $\delta = 0$ means that only this kind of odd behaviour can avoid Bad.

For $\alpha > 1$, there is a non-singular range for δ. The profile of Figure 4.4 is established by HyTech and it can be intuitively justified as follows. First, observe that the two clocks x and w are reset synchronously and thus we have $w = x$ for all executions of the system. Since the location Bad is reachable whenever x is above α, the worst situation corresponds to the largest value of x (and w). This occurs when the delays in the controller are the largest: let us jump to ℓ'_2 when $w = 1 - \delta$ and count the time needed to get back to ℓ'_1. First, we wait δ unit of time in ℓ'_2 for the event B! to be issued, then one more δ in location ℓ'_2 to see the event and finally in ℓ'_3, a third δ is allowed to react and output C. At the end, we have $w = 3\delta$. On the other hand, we can wait in ℓ'_1 until $w = 1 + \delta$ to issue an A again. Thus, to avoid the location Bad, we need the condition $\delta \leq \min\{\alpha - 1, \frac{\alpha}{3}\}$ for ensuring implementability.

In practice, we must choose a hardware such that $3\Delta_L + 2\Delta_P \leq \delta$ (according to Theorem 4.20). For example, if $\alpha \geq 2$ we can take $3\Delta_L + 2\Delta_P = \frac{2}{3}$.

4.5 Related Works

Research in real-time systems has early focused on the questions related to the decidability of the formalisms for real-time, see for example [Koy90, AH92, AD94, AFH96, HKPV98]. More recently, there are several works that study the relationship between high-level models and low-level implementations. We give an overview of the main results for real-time systems. They were all published in the last decade and most of them in the years 2000.

4.5.0.1 PLC

To our knowledge, the work on PLC-automata [Die99, Die01] is the closest to the AASAP semantics. It is one of the first attempts to formally prove the correctness of a real-time implementation with the imperfections of the hardware taken into account. The motivation was to verify real-time applications for Programmable Logic Controllers (PLC). The approach proposed by Dierks compares with ours as follows. Starting from the observation that the classical semantics of real-time systems neglects the delay for communications and computations, a new and weaker semantics for PLC is given, the PLC-automata. While being dedicated to physical PLC, the model intends to cover the main characteristics of all real-time operating systems, namely the cycle (*i*) *polling* of the inputs, (*ii*) *computing* the reactions, and (*iii*) *updating* the outputs. The components of a PLC-automaton include an *upper time bound* for a cycle, *delays* for the visibility of inputs, and *accuracy* of timers. This methodology is similar to our approach with the program semantics of Section 4.3.1. Dierks goes one step forward and provides an algorithm for translating PLC-automata to code, but the correctness is not established formally.

The language used to define the semantics of PLC-automata and to specify real-time constraints is the Duration Calculus [CHR91] which is a dense time interval-based temporal logics that allows to fix the delays for the visibility of events, the computation times, the imprecision of the clocks, etc. That logics is very expressive, but its semantics may appear relatively obscure for the novice as it relies on the integral calculus, which is a source of counter-intuitive interpretation, such as for example "a proposition is true *almost always*", that is at every point in time except at most finitely many. The use of subtle tricks is necessary to circumvent this kind of unexpected meaning. More pragmatically, the major drawback is that the Duration Calculus is undecidable [CHS93]. By contrast, we show in Chapter 6 that the question of implementability is decidable with the AASAP semantics. Moreover, our semantics is more abstract in that it does not make explicit the different kinds of imprecisions of the implementation. The semantics of PLC-automata is more similar to the program semantics.

In summary, the framework of PLC-automata provides an interesting solution for the design of real-time controllers that are aware of the characteristics of the target platform. It is undoubtedly an important step in bridging the gap between models and implementations in real-time.

4.5.0.2 Giotto

Giotto is a programming environment to develop embedded control systems [HHK01]. It is used to help the implementation of high-level mathematical designs by software engineers. The main feature of Giotto is an abstract programming language, that is (*i*) closer to the design level than classical programming languages, which makes easier

for designers the translation from mathematical model to code, and (*ii*) suitable to formal verification for checking correctness, and such that code generation is possible for potentially any platform (that is the combination of a hardware and a real-time operating system).

The compilation to a particular target platform follows two steps. First, the abstract program is translated to E-code, a language that is executed by a virtual machine, the E-machine [HK02]. The abstract program describes the structure of the *tasks* (decomposed in modes of operation, with logical execution times) and their interaction with each other through I/O *drivers*. The corresponding E-code is a set of blocks that execute a sequence of primitive instructions similar to assembly languages. For going to executable code, it is possible to specify some constraints about the target platform, such as the number of CPUs, the worst case execution times of the tasks, the internal communication delays, and a jitter tolerance for the writing of outputs. Moreover, GIOTTO is not restricted to a particular scheduling policy; the user can make his choice. Several compilers for GIOTTO have been developed, and realistic applications have been implemented [San99].

To make a comparison, we could say that the AASAP semantics gives conditions for implementability on a time-triggered architecture of a controller that is event-triggered, while GIOTTO starts from a model that is already time-triggered. That approach is closer to synchronous languages like ESTEREL [BG92] and LUSTRE [HCRP91].

4.5.0.3 CHARON

CHARON is a design toolkit for hybrid systems. A hybrid system is described in terms of several *agents* that communicate via shared variables. Inside an agent, the behaviour is decomposed in *modes* (similar to the locations of an hybrid automaton) that are connected by edges labelled by guarded commands. In each mode, the evolution of the variables can be described by differential equations. The language allows a modular design in modes that can have sub-modes, and can be shared by other modes. CHARON has a continuous-time semantics and it supports formal verification of safety properties, based on reachability analysis [LPY01].

A code generator has been developed for CHARON, that maps models with several (concurrent) agents to (sequential) code for a single processor. Therefore, the translation has to take care of the dependencies between the shared variables. On the other hand, the code is by essence discrete, while the model is continuous. The problems related to the discretization of hybrid systems are carefully studied and proofs are given that the code refines (or equivalently is simulable by) the model [AIK+03, HKLC04].

The methodology has been applied to several case studies, and seems to be promising for the model-based development of embedded systems. However, some aspects that are important for the implementability, such as the delays in sensing/actuating, and the existence of non null computation times have been left as future works. Maybe

interesting synergies can be found with the AASAP semantics.

4.5.0.4 Other works

In [AFM+02, AFP+03], Yi *et al.* have developed the tool TIMES that generates executable code (C code for the LEGO MINDSTORMS™ platform) from timed automata with an explicit model of the computation *tasks* (the automaton releases task with given deadlines and worst-case execution times). The tool integrates a schedulability analysis of the tasks, but the code is generated with the synchrony hypothesis. Therefore, the properties that are proven on the models are not guaranteed to be preserved by their code generation.

In [IKL+00], Larsen *et al.* show how to translate code for real-time controllers to timed automata in UPPAAL to formally verify the correctness of the implementation. Usually, they encounter the problem that the obtained timed automata are difficult to analyze because the time unit at the controller level (the time slice of the real-time OS for example) is much smaller than the time unit of the environment. This leads to what they call *symbolic state space fragmentation*. They proposed in [HL02] a partial solution to that problem, namely accelerating some cycles that have a particularly simple form. In our framework, we do not encounter that problem because the reaction delay of the AASAP semantics is usually close to the time unit of the environment to control, and usually much larger than the time unit of the hardware on which the control program is executed. The advantage of our approach is also that Theorem 4.20 and Corollary 4.21 hold for all ELASTIC controllers, and thus we do not need to redo a proof of correctness for each application.

In [AT05], Altisen and Tripakis attack the problem of implementability with a methodology where the target platform has to be explicitly modeled as a timed automaton. The controller is transformed into a finite (untimed) automaton that is *triggered* by a network of timed automata that model the features of the execution platform such as the precision of digital clocks, the communication between the program and the environment, the scheduling policy etc. This way, the methodology is more flexible and more expressive as the designer can easily change some characteristics of the target platform. However, the approach suffers from the common problem that *faster* is not necessarily *better* and the model-checking may be difficult because the models contain all the details of the implementation.

Chapter 5

Verification of the AASAP Semantics

> Ouvrez des écoles, vous fermerez des prisons.
>
> Victor Hugo.

5.1 Introduction

In this chapter, we present a proof that the AASAP semantics can be verified algorithmically when the parameter δ is a *fixed* rational number. The question of deciding the existence of δ is studied in Chapter 6.

Formally, we show that the [Fixed] flavour of the robust safety control problem is decidable. To do this, we encode the AASAP semantics of an ELASTIC controller with an Alur-Dill automaton such that the two systems are mutually similar [DDR04]. The encoding is generic for all values of δ. We need to fix δ only to obtain a non-parametric timed automaton that can be verified algorithmically. This technique is also useful in practice to decide the robust correctness of a controller, since we can use the recent and powerful tools for the analysis of timed automata, like UPPAAL and KRONOS. Even, in the case where δ is left as a parameter, we can use tools for hybrid automata like HYTECH and PHAVER. However, the encoding we proposed in [DDR04] has a limited interest in practice because its size is exponential in $|\mathsf{Lab}_{\mathsf{in}}|$, the number of input labels of the controller, essentially because we need a queue of size 1 for each input event, thus at least $2^{|\mathsf{Lab}_{\mathsf{in}}|}$ discrete configurations.

We solved this problem by giving an encoding which is compositional in the sense that we distribute the queues of each event in a separate automaton. This way, the encoding has polynomial size. Of course, if we compute the synchronized product of our construction, we get back to an exponential size. However, this is not a drawback in practice because networks of timed automata can be analyzed on-the-fly without primarily computing a product, and it is likely in practice that a minor subset of the $2^{|\mathsf{Lab}_{\mathsf{in}}|}$ configurations is reachable. Therefore, we avoid to explore the whole state space

and we obtain a significant increase of performance, as shown by experiments.

So, we want to encode the AASAP semantics (parameterized by Δ) in a compositional way, that is with a network of timed automata. This approach leads to technical difficulties related to the distributed form of the encoding. In particular, to decide when a transition becomes urgent is no more possible within one automaton because several components are involved. For example, the time spent in a location and the recording of input events are distributed over different automata. In Section 5.2, we show how to deal with urgency in a compositional way. In Section 5.3, we describe two components of the compositional encoding, called the *watchers* that are used to distribute the recording of the timing information about input events. The complete construction is given in Section 5.4 and a proof of correctness is given in Section 5.5.

5.2 Urgency policies

In this section, we define a syntactical feature that is used to model urgency, the flag Asap. We will see that in the case of a product of timed automata, the meaning of that flag cannot be encoded using only clocks. This is not surprising since the purpose of urgency policies is to declare urgent an edge in a product of timed automata whenever it is urgent in *every* automaton of the product. With clocks, urgency is local and so an edge would be urgent when it is urgent in *some* automaton of the product.

Definition 5.1 [Urgency policy] An *urgency policy* for a timed automaton A with set of edges Edg_A is a boolean function $\mathsf{Asap} : \mathsf{Edg}_A \to \{\top, \bot\}$. □

We use the semantics of the tool HyTech for the Asap flag [HHW95]. The flag modifies the condition for the passage of time: time can pass in a location if there is no outgoing urgent edge labelled by Asap. This may be counter-intuitive as time is blocked even when the guards of the urgent edge are not satisfied. Notice that a slightly different notion of urgency is available in Uppaal, which means that some adaptations are needed to define an encoding of the AASAP semantics that is compatible with Uppaal.

Definition 5.2 [Semantics of urgency policies] The *semantics* of a timed automaton A with urgency policy defined by Asap is the TTS $\langle Q, Q_0, Q_f, \Sigma, \to' \rangle$ such that $[\![A]\!] = \langle Q, Q_0, Q_f, \Sigma, \to \rangle$ and $((\ell, v), \sigma, (\ell', v')) \in \to'$ if and only if $((\ell, v), \sigma, (\ell', v')) \in \to$ and either $\sigma \in \Sigma$, or $\sigma \in \mathbb{T}$ and for every edge $e = (\ell_1, \ell_2, g, \sigma, R) \in \mathsf{Edg}_A$ of A, if $\ell_1 = \ell$ then $\mathsf{Asap}(e) = \bot$. □

By an abuse of notation, we denote by $[\![A]\!]$ the semantics of Definition 5.2. Notice that the semantics of classical timed automata (Definition 2.12) is equal to the semantics of timed automata with the urgency policy $\mathsf{Asap}_\bot$ that maps every edge to $\bot$.

This explains why we use the same notation. Further, the urgency policy in a single timed automaton can be encoded using one additional clock as follows: let us call a location ℓ *urgent* in A (noted $\mathsf{Urgent}_A(\ell)$) whenever there exists an edge e of A with source location ℓ and $\mathsf{Asap}(e) = \top$. Now, we use a new clock u to ask that no time can pass in urgent locations. We add the predicate $u \leq 0$ conjunctively in the invariant of each of them, and we reset the clock u on every edge. Hence, this shows that the urgency policy does not add expressive power to timed automata. It is very useful however when used in combination with synchronized products to model urgency in a compositional way. For the synchronized products of timed automata with urgency policy, we extend Definition 2.18 as follows:

Definition 5.3 The *synchronized product* of two timed automata $A_1 = \langle \mathsf{Loc}^1, \mathsf{Var}^1, \mathsf{Init}^1, \mathsf{Inv}^1, \mathsf{Lab}^1, \mathsf{Edg}^1, \mathsf{Final}^1 \rangle$ and $A_2 = \langle \mathsf{Loc}^2, \mathsf{Var}^2, \mathsf{Init}^2, \mathsf{Inv}^2, \mathsf{Lab}^2, \mathsf{Edg}^2, \mathsf{Final}^2 \rangle$ on $\mathcal{G}$ with respective urgency policies Asap_1 and Asap_2 is the timed automaton $A_1 \times A_2 = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ on $\mathcal{G}$ with urgency policy Asap such that:

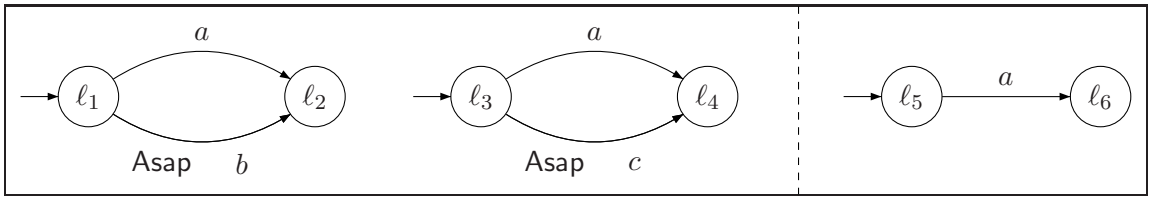
- $\mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Final}$ are defined as in Definition 2.18;
- An edge $e = ((\ell_1, \ell_2), (\ell'_1, \ell'_2), g, \sigma, R) \in \mathsf{Edg}$ iff one of the following assertions holds:
  - $e_1 = (\ell_1, \ell'_1, g_1, \sigma, R_1) \in \mathsf{Edg}^1$, $e_2 = (\ell_2, \ell'_2, g_2, \sigma, R_2) \in \mathsf{Edg}^2$, $\sigma \neq \tau$, $g = g_1 \wedge g_2$ and $R = R_1 \cup R_2$. Moreover, we have $\mathsf{Asap}(e) = \top$ iff $\mathsf{Asap}_1(e_1) = \top$ or $\mathsf{Asap}_2(e_2) = \top$.
  - $e_1 = (\ell_1, \ell'_1, g, \sigma, R) \in \mathsf{Edg}^1$, $\ell_2 = \ell'_2$ and $\sigma = \tau$ or $\sigma \notin \mathsf{Lab}^2$. We have $\mathsf{Asap}(e) = \mathsf{Asap}_1(e_1)$.
  - $\ell_1 = \ell'_1$, $\sigma = \tau$ or $\sigma \notin \mathsf{Lab}^1$, and $e_2 = (\ell_2, \ell'_2, g, \sigma, R) \in \mathsf{Edg}^2$. We have $\mathsf{Asap}(e) = \mathsf{Asap}_2(e_2)$.

□

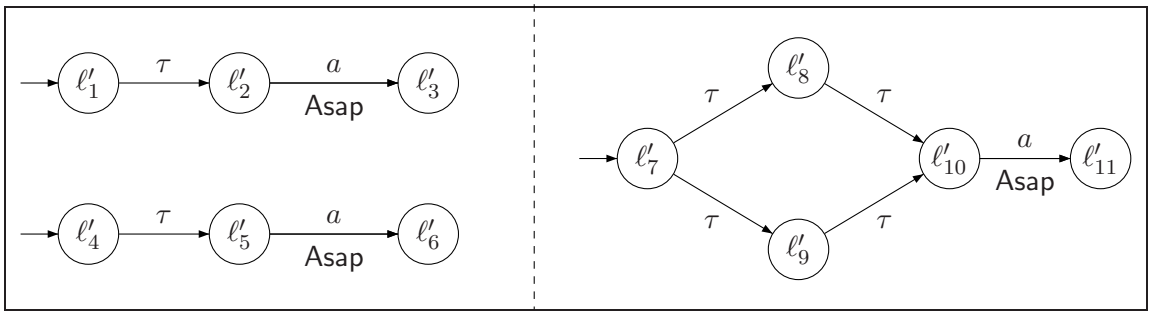
The reader can check that the encoding of urgency sketched above is not preserved by synchronized product since $\mathsf{Urgent}_{A_1 \times A_2}(\ell_1, \ell_2) \neq \mathsf{Urgent}_{A_1}(\ell_1) \vee \mathsf{Urgent}_{A_2}(\ell_2)$. In fact, we cannot define $\mathsf{Urgent}_{A_1 \times A_2}(\ell_1, \ell_2)$ as a boolean combination of $\mathsf{Urgent}_{A_1}(\ell_1)$ and $\mathsf{Urgent}_{A_2}(\ell_2)$, as shown on Figure 5.1(a) for conjunction and Figure 5.1(b) for disjunction. The figures show two automata on the left part and their synchronized product on the right part. In Figure 5.1(a), we assume that the alphabet of A_1 and A_2 is $\{a, b, c\}$.

5.3 Watchers

For the rest of this section, we fix a *parameter* Δ that we distinguish from its value $\delta \in \mathbb{Q}$. We define parametric predicates that use only that single parameter.



(a) $\mathsf{Urgent}(\ell_5) \neq \mathsf{Urgent}(\ell_1) \wedge \mathsf{Urgent}(\ell_3)$.



(b) $\mathsf{Urgent}(\ell'_8) \neq \mathsf{Urgent}(\ell'_2) \vee \mathsf{Urgent}(\ell'_4)$.

Figure 5.1: Urgency and composition.

Definition 5.4 [Parametric multirectangular predicates] Given a set Var of clocks, a *parametric multirectangular predicate* over Var is a finite formula φ defined by the following grammar rule:

$$\varphi ::= \bot \mid \top \mid x \bowtie a \mid \varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

where $x \in \mathsf{Var}$, $a \in \{c, c+\Delta, c-\Delta \mid c \in \mathbb{N}\}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$.

We denote by MultPRect the class of parametric multirectangular predicates.

□

A multirectangle is a finite union of rectangles. Given $\delta \in \mathbb{Q}^{\geq 0}$, the definition of the satisfiability relation $v \models_\delta \varphi$ for a valuation $v : \mathsf{Var} \to \mathbb{T}$ and a parametric rectangular predicate $\varphi \in \mathsf{MultPRect}$ is similar to $v \models_\kappa \varphi$ with $\kappa(\Delta) = \delta$ in Definition 3.2. Similarly, for $\delta \in \mathbb{Q}^{\geq 0}$ the semantics of a timed automaton A over MultPRect is induced by $\models_\delta$ and is denoted $[\![A]\!]_\delta$.

The main characteristics of the AASAP semantics that is difficult to encode compositionally is expressed by rule $(A5.1)$ and $(A5.2)$ in Definition 4.15 that defines the *almost urgency*. According to that rule, there are three reasons for allowing time to pass:

1. either the controller has been in its current location for less than δ time units,

2. or all the pending input events have been issued by the environment less than δ time units ago,

3. or finally the guards of the outgoing transitions have not been enabled for more than δ time units.

Roughly, those conditions will be checked in our compositional construction by, respectively, a syntactical transformation $\mathcal{E}(A)$ of the ELASTIC controller A, and two types of widgets: the *event-watchers* and the *guard-watchers*.

The most obvious way of defining urgency with timed automata is by using invariants on locations. Roughly, if we have an edge guarded by a constraint $g \equiv x \geq c$, it can be forced as soon as it is enabled by adding as invariant in its source location the closure of $\neg g$, that is $x \leq c$. This way, time is blocked when the guard is satisfied and the discrete transition is forced. If we enlarge the invariant by Δ (yielding $x \leq c + \Delta$), we get the *almost urgency*. However, urgency is not only determined by guards, but also by the discrete events: the location changes and the input events. For the edge guarded by g, a valid behaviour would be that the source location is reached (or the input is received) when $x = c + 1$, although the invariant is then violated. Hence, we have to take into account the conditions of urgency altogether according to rule $(A.5)$ of the AASAP semantics.

We introduce the following notations to simplify the presentation of the construction. It is based on Definition 4.11.

Definition 5.5 Let $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ be an ELASTIC controller structured by $(\mathsf{Lab}_{\mathsf{in}}, \mathsf{Lab}_{\mathsf{out}}, \mathsf{Lab}_{\tau})$, and $\ell \in \mathsf{Loc}$ be one of its locations. Define the set of guards labelling output transitions or internal transitions as:

$$G_{\mathsf{act}}(\ell) = \{g \mid (\ell, \ell', g, \sigma, R) \in \mathsf{Edg} \wedge \sigma \in \mathsf{Lab}_{\mathsf{out}} \cup \mathsf{Lab}_{\tau}\}$$

For each $\alpha \in \mathsf{Lab}_{\mathsf{in}}$, define the set of guards labelling α-transitions as:

$$G_{\mathsf{evt}}(\ell, \alpha) = \{g \mid (\ell, \ell', g, \alpha, R) \in \mathsf{Edg}\}$$

Finally, we define the following parametric multirectangular predicates:

$$\bar{\varphi}_{\mathsf{act}}(\ell) = \bigwedge_{g \in G_{\mathsf{act}}(\ell)} \neg(_{-\Delta}]g[_0) \qquad \text{and} \qquad \bar{\varphi}_{\mathsf{evt}}(\ell, \alpha) = \bigwedge_{g \in G_{\mathsf{evt}}(\ell, \alpha)} \neg(_{-\Delta}]g[_0)$$

(with the usual convention that $\bigwedge_{g \in \varnothing} \varphi_g \equiv \top$).

□

E.g. for $G_{\mathsf{act}}(l) = \{x \geq 3, 0 \leq y \leq 1\}$, we have $\bar{\varphi}_{\mathsf{act}}(l) \equiv x \leq 3 + \Delta \wedge (y \leq \Delta \vee y \geq 1)$.

The predicates $\bar{\varphi}_{\mathsf{act}}(\ell)$ and $\bar{\varphi}_{\mathsf{evt}}(\ell, \alpha)$ are used as invariants in $\mathcal{E}(A)$ and the *guard-watchers*, to check the second part of rule $(A5.1)$ and the third part of rule $(A5.2)$ respectively. We will see in Lemma 5.12 that there is a strong link between the set $\{v \mid v \models_\delta \neg(_{-\Delta}]g[_0)\}$ and the set $\{v \mid \mathsf{ETime}(g, v) \leq \delta\}$ (they are almost equal, namely their closure are equal).

Those invariants are central to our construction, but if we want a compositional construction (a product of automata), invariants are too restrictive to express urgency since urgency also depends on the current state of the other automata offering enabled synchronizations in the product. Hence, we should not block time simply when a transition is enabled in *one* automaton but only when it is enabled in *every* automaton of the product. Therefore, some compositional mechanism is needed to model urgency in a product: we will use the Asap flag. Remember that this flag expresses the fact that a transition is urgent as soon as it is enabled in the product.

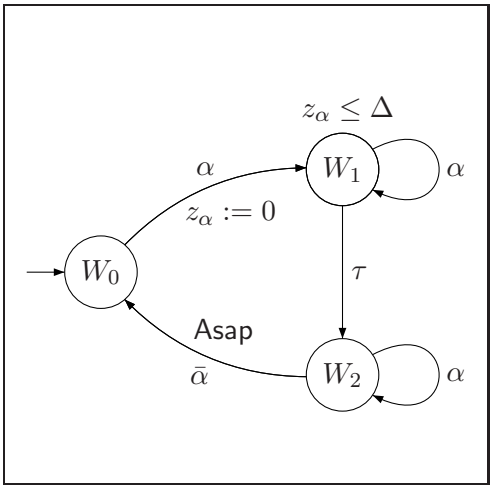


Figure 5.2: Event-Watcher W_α.

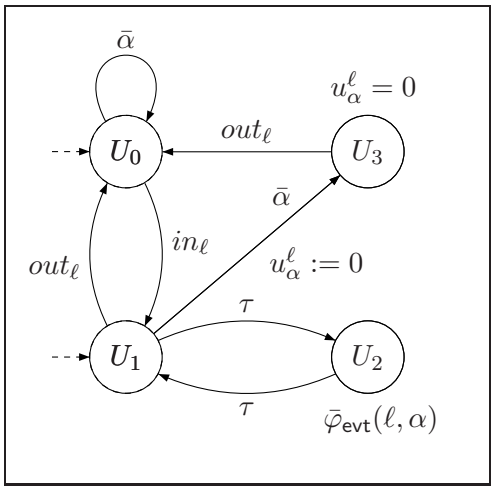


Figure 5.3: Guard-Watcher W^ℓ_α.

Now, we present the two types of widgets that are used in the compositional construction for the AASAP semantics of an ELASTIC controller $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ structured by $(\mathsf{Lab}_{\mathsf{in}}, \mathsf{Lab}_{\mathsf{out}}, \mathsf{Lab}_\tau)$.

5.3.1 Event-watchers

Associated to each event $\alpha \in \mathsf{Lab}_{\mathsf{in}}$, we define W_α (see Figure 5.2) that records the occurrences of the event α. It has a clock z_α that encodes the value of $I(\alpha)$ from the AASAP semantics. The clock z_α records the time elapsed since the last untreated event α was issued by the environment. Thus when $I(\alpha) \neq \bot$, the value of the clock z_α is equal to $I(\alpha)$ and the location of the automaton is either W_1 or W_2. According to rule $(A2)$ in Definition 4.15, in those two states, the event α is ignored (the self-loops guarantee input enabledness). The automaton can stay in location W_1 as long

as $z_\alpha \leq \Delta$. After that delay at the latest, the location moves to W_2 where a transition labelled by $\bar{\alpha}$ is enabled, and possibly urgent due to the urgency policy.

Definition 5.6 [Event-watcher] Let $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ be an E-LASTIC controller structured by $(\mathsf{Lab_{in}}, \mathsf{Lab_{out}}, \mathsf{Lab}_\tau)$ and $\alpha \in \mathsf{Lab_{in}}$ an input event. Define the timed automaton *event-watcher* $W_\alpha = \langle \mathsf{Loc}', \mathsf{Var}', \mathsf{Init}', \mathsf{Inv}', \mathsf{Lab}', \mathsf{Edg}', \mathsf{Final}' \rangle$ and its urgency policy Asap as follows:

- $\mathsf{Loc}' = \{W_0, W_1, W_2\}$;
- $\mathsf{Var}' = \{z_\alpha\}$;
- $\mathsf{Init}'(W_0) \equiv z_\alpha = 0$ and $\mathsf{Init}'(W_1) \equiv \mathsf{Init}'(W_2) \equiv \bot$;
- $\mathsf{Inv}'(W_0) \equiv \mathsf{Inv}'(W_2) \equiv \top$ and $\mathsf{Inv}'(W_1) \equiv z_\alpha \leq \Delta$;
- $\mathsf{Lab}' = \{\alpha, \bar{\alpha}, \tau\}$;
- $\mathsf{Edg}' = \{e_1, e_2, e_3, e_4, e_5\}$ where:
  - $e_1 = (W_0, W_1, \top, \alpha, \{z_\alpha\})$,
  - $e_2 = (W_1, W_2, \top, \tau, \varnothing)$,
  - $e_3 = (W_2, W_0, \top, \bar{\alpha}, \varnothing)$,
  - $e_4 = (W_1, W_1, \top, \alpha, \varnothing)$,
  - $e_5 = (W_2, W_2, \top, \alpha, \varnothing)$;
- $\mathsf{Final}'(\ell) = \bot$ for every $\ell \in \mathsf{Loc}'$;
- $\mathsf{Asap}(e_3) = \top$ and $\mathsf{Asap}(e) = \bot$ for every $e \in \mathsf{Edg}' \setminus \{e_3\}$.

□

5.3.2 Guard-watchers

Associated to each event $\alpha \in \mathsf{Lab_{in}}$ and location $\ell \in \mathsf{Loc}$, we define W_α^ℓ (see Figure 5.3) to monitor the truth value of the set of guards $G = \bar{\varphi}_{\mathsf{evt}}(\ell, \alpha)$. When the ELASTIC controller A is not in location ℓ, the guard-watchers W_α^ℓ do not influence the execution, being in location U_0 and offering a self-loop synchronization on $\bar{\alpha}$. When the location ℓ is reached in A, a synchronization on in_ℓ occurs and forces each W_α^ℓ to enter their location U_1 and to become active. The guard-watchers get back in U_0 as soon as ℓ is exited, through the label out_ℓ. Thus, the guard-watchers are active when they are not in location U_0. Their role is then to switch between locations U_1 and U_2 depending on the truth value of the conditions $\mathsf{ETime}(g, v) \leq \delta$ that appears in rule $(A5.2)$ of

Definition 4.15, where g is a guard on a transition of A labelled with α (see also the comments after Definition 5.5). When the condition is not true (in location U_1), a transition labelled by $\bar{\alpha}$ is enabled. When this transition is taken, the location U_3 is visited and left immediately (because of the invariant $u_\alpha^\ell = 0$) so that the controller leaves location ℓ immediately.

Remember that there is only one location ℓ in an ELASTIC controller such that $\mathsf{Init}(\ell)$ is satisfiable (Definition 4.9). The initial location of the corresponding guard-watchers W_α^ℓ is U_1.

Definition 5.7 [Guard-watcher] Let $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ be an ELASTIC controller structured by $(\mathsf{Lab}_{\mathsf{in}}, \mathsf{Lab}_{\mathsf{out}}, \mathsf{Lab}_\tau)$, let $\ell \in \mathsf{Loc}$ be one of its locations and $\alpha \in \mathsf{Lab}_{\mathsf{in}}$ an input event. Let $G \subseteq \mathsf{Rect}(\mathsf{Var})$ be a set of guards. Define the timed automaton *guard-watcher* $W_\alpha^\ell = \langle \mathsf{Loc}', \mathsf{Var}', \mathsf{Init}', \mathsf{Inv}', \mathsf{Lab}', \mathsf{Edg}', \mathsf{Final}' \rangle$ and its urgency policy Asap as follows:

- $\mathsf{Loc}' = \{U_0, U_1, U_2, U_3\}$;
- $\mathsf{Var}' = \mathsf{Var} \cup \{u_\alpha^\ell\}$ (assuming $u_\alpha^\ell \notin \mathsf{Var}$);
- $\mathsf{Init}'(U_{\mathsf{init}}) \equiv \top$ and $\mathsf{Init}'(U) \equiv \bot$ for every $U \in \mathsf{Loc} \setminus \{U_{\mathsf{init}}\}$ where $U_{\mathsf{init}} = U_1$ if $\mathsf{Init}(\ell)$ is satisfiable and $U_{\mathsf{init}} = U_0$ otherwise;
- $\mathsf{Inv}'(U_0) \equiv \mathsf{Inv}'(U_1) \equiv \top$, $\mathsf{Inv}'(U_2) \equiv \bar{\varphi}_{\mathsf{evt}}(\ell, \alpha)$ and $\mathsf{Inv}'(U_3) \equiv u_\alpha^\ell = 0$;
- $\mathsf{Lab}' = \{\bar{\alpha}, in_\ell, out_\ell, \tau\}$;
- $\mathsf{Edg}' = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ where:
  - $e_1 = (U_0, U_1, \top, in_\ell, \varnothing)$,
  - $e_2 = (U_1, U_0, \top, out_\ell, \varnothing)$,
  - $e_3 = (U_1, U_2, \top, \tau, \varnothing)$,
  - $e_4 = (U_2, U_1, \top, \tau, \varnothing)$,
  - $e_5 = (U_1, U_3, \top, \bar{\alpha}, \{u_\alpha^\ell\})$;
  - $e_6 = (U_3, U_0, \top, out_\ell, \varnothing)$;
  - $e_7 = (U_0, U_0, \top, \bar{\alpha}, \varnothing)$;
- $\mathsf{Final}'(\ell) = \bot$ for every $\ell \in \mathsf{Loc}'$;
- $\mathsf{Asap}(e) = \bot$ for every $e \in \mathsf{Edg}'$.

□

5.4 Compositional Construction

Now, we present the syntactical transformation $\mathcal{E}(\cdot)$ of ELASTIC controllers. The aim of $\mathcal{E}(A)$ is of course to encode the structure of A, but also to translate the rules of the AASAP semantics (Definition 4.15) that have not been encoded in the watchers. The transformation is similar for each location of the controller, so we illustrate it on location ℓ of the ELASTIC controller A of Figure 5.4. The result of the transformation is shown on Figure 5.5:

- From the first part of rules $(A5.1)$ and $(A5.2)$, we must allow a maximum delay of Δ when the location changes. When the controller A moves to the location ℓ, the automaton $\mathcal{E}(A)$ enters location In_ℓ with $d = 0$. The invariant forces a move to Out_ℓ before Δ time units.
- From the second part of rule $(A5.1)$, transitions labeled with actions $\sigma \in \mathsf{Lab}_{\mathsf{out}} \cup \mathsf{Lab}_\tau$ should be urgent when their guard has been satisfied for more than Δ time units. This is encoded in the invariant $d \leq \Delta \vee \bar{\varphi}_{\mathsf{act}}(\ell)$ of location Out_ℓ. Notice that the second part of rule $(A5.2)$ is encoded by watchers.
- From rules $(A1)$ and $(A3)$, we have to enlarge the guards of the controller's transitions.

Finally, when a transition from A is fired, the location Out_ℓ is left and the two events out_ℓ and $in_{\ell'}$ are issued where ℓ' is the target location of the transition.

Definition 5.8 [Controller transformation $\mathcal{E}$] Let $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ be an ELASTIC controller structured by $(\mathsf{Lab}_{\mathsf{in}}, \mathsf{Lab}_{\mathsf{out}}, \mathsf{Lab}_\tau)$. Define the timed automaton $\mathcal{E}(A) = \langle \mathsf{Loc}', \mathsf{Var}', \mathsf{Init}', \mathsf{Inv}', \mathsf{Lab}', \mathsf{Edg}', \mathsf{Final}' \rangle$ and its urgency policy Asap as follows:

- $\mathsf{Loc}' = \{PreIn_\ell, In_\ell, Out_\ell, PostOut_{\ell,\ell'} \mid \ell, \ell' \in Loc\}$;
- $\mathsf{Var}' = \mathsf{Var} \cup \{d\}$ (assuming $d \notin \mathsf{Var}$);
- $\mathsf{Init}'(In_\ell) \equiv \mathsf{Init}(\ell) \wedge d = 0$ and $\mathsf{Init}'(PreIn_\ell) \equiv \mathsf{Init}'(Out_\ell) \equiv \mathsf{Init}'(PostOut_{\ell,\ell'}) \equiv \bot$ for every $\ell, \ell' \in Loc$;
- Inv' is defined as follows for each $\ell, \ell' \in \mathsf{Loc}$:
  - $\mathsf{Inv}'(In_\ell) \equiv d \leq \Delta$,
  - $\mathsf{Inv}'(Out_\ell) \equiv d \leq \Delta \vee \bar{\varphi}_{\mathsf{act}}(\ell)$,
  - $\mathsf{Inv}'(PreIn_\ell) \equiv \mathsf{Inv}'(PostOut_{\ell,\ell'}) \equiv d = 0$;
- $\mathsf{Lab}' = \mathsf{Lab}_{\mathsf{out}} \cup \mathsf{Lab}_\tau \cup \overline{\mathsf{Lab}_{\mathsf{in}}} \cup \bigcup_{\ell \in \mathsf{Loc}} \{in_\ell, out_\ell\}$ (assuming those sets are disjoint);

- Edg' contains the edges $(Out_\ell, PostOut_{\ell,\ell'}, {}_\Delta[g]_\Delta, \sigma, R \cup \{d\})$ such that either
  - there exists $(\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab_{out}} \cup \mathsf{Lab}_\tau$,
  - or there exists $(\ell, \ell', g, \alpha, R) \in \mathsf{Edg}$ with $\alpha \in \mathsf{Lab_{in}}$ and $\sigma = \bar{\alpha}$;

  and the edges $(In_\ell, Out_\ell, \top, \tau, \varnothing)$, $(PostOut_{\ell,\ell'}, PreIn_{\ell'}, \top, out_\ell, \varnothing)$ and $(PreIn_\ell, In_\ell, \top, in_\ell, \varnothing)$ for each $\ell, \ell' \in Loc$;
- $\mathsf{Final}'(\ell^{\mathcal{E}}) = \bot$ for every $\ell^{\mathcal{E}} \in \mathsf{Loc}'$;
- $\mathsf{Asap}(e) = \bot$ for every $e \in \mathsf{Edg}'$.

□

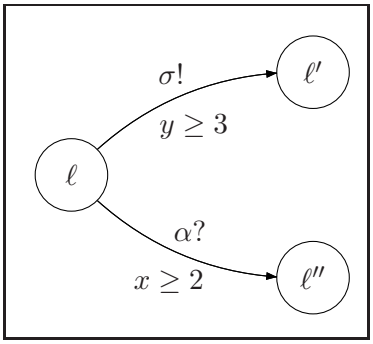


Figure 5.4: An ELASTIC controller A.

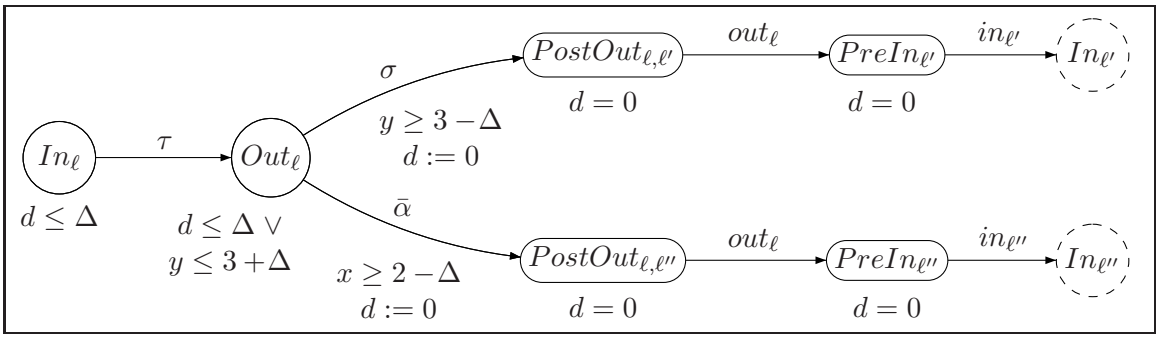


Figure 5.5: The timed automaton $\mathcal{E}(A)$ associated to the ELASTIC controller A of Figure 5.4.

Running example Figure 5.6 shows the result of the transformation $\mathcal{E}(\cdot)$ of Definition 5.8 when applied to the controller of Figure 4.1(b). Remember that only one synchronization label is allowed per edge, as usual in the literature and in model-checkers. For the sake of clarity however, we depict by a single edge, the sequences of three simultaneous (but sequential) synchronizations $\sigma, out_\ell, in_\ell$. Using this encoding, we have obtained the graph of Figure 4.4 with HYTECH.

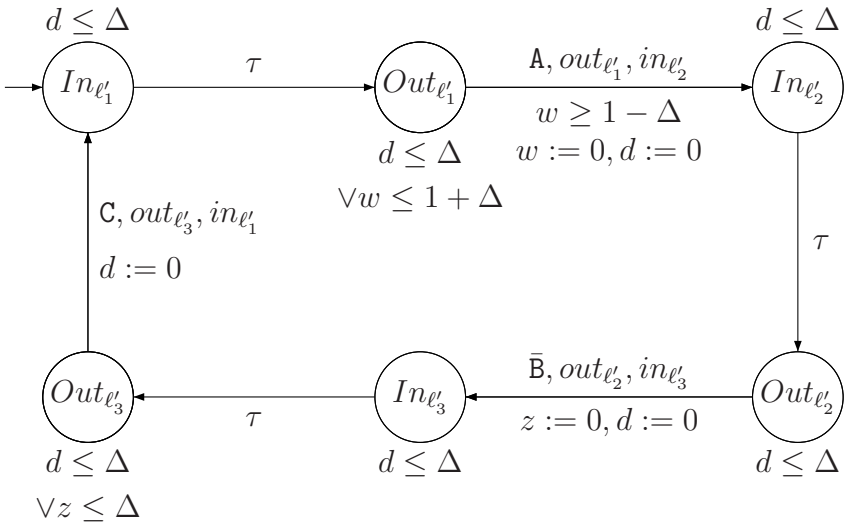


Figure 5.6: Running example: controller transformation.

Definition 5.9 [Compositional construction $\mathcal{F}$] Let $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ be an ELASTIC controller structured by $(\mathsf{Lab}_{\mathsf{in}}, \mathsf{Lab}_{\mathsf{out}}, \mathsf{Lab}_{\tau})$. The *compositional construction* $\mathcal{F}(A)$ is the synchronized product of the following timed automata:

- the event-watchers W_α for every $\alpha \in \mathsf{Lab}_{\mathsf{in}}$,
- the guard-watchers W^ℓ_α for every $\alpha \in \mathsf{Lab}_{\mathsf{in}}$ and $\ell \in \mathsf{Loc}$,
- and the timed automaton $\mathcal{E}(A)$.

Formally, $\mathcal{F}(A) = \mathcal{E}(A) \times \prod_{\alpha \in \mathsf{Lab}_{\mathsf{in}}} W_\alpha \times \prod_{\alpha \in \mathsf{Lab}_{\mathsf{in}}, \ell \in \mathsf{Loc}} W^\ell_\alpha$.

Given $\delta \in \mathbb{Q}^{\geq 0}$, the *compositional semantics* of A is the STTS $[\![A]\!]^{\mathcal{F}}_\delta$ given by $[\![\mathcal{F}(A)]\!]_\delta[\{in_\ell, out_\ell\} := \tau]$ structured by $(\mathsf{Lab}_{\mathsf{in}}, \mathsf{Lab}_{\mathsf{out}}, \mathsf{Lab}_\tau \cup \overline{\mathsf{Lab}_{\mathsf{in}}} \cup \{\tau\})$. □

5.5 Correctness proof

The correctness of the compositional construction is established by a proof of mutual simulation with the AASAP semantics.

Theorem 5.10 *Let A be an* ELASTIC *controller embedded in an* STTS *environment* Env. *For all $\delta \in \mathbb{Q}^{\geq 0}$ we have $[\![A]\!]^{\mathsf{AAsap}}_\delta \succeq_{weak} [\![A]\!]^{\mathcal{F}}_\delta$ and $[\![A]\!]^{\mathcal{F}}_\delta \succeq_{weak} [\![A]\!]^{\mathsf{AAsap}}_\delta$.*

The proof of this theorem is given after the following intermediate definitions and lemmas.

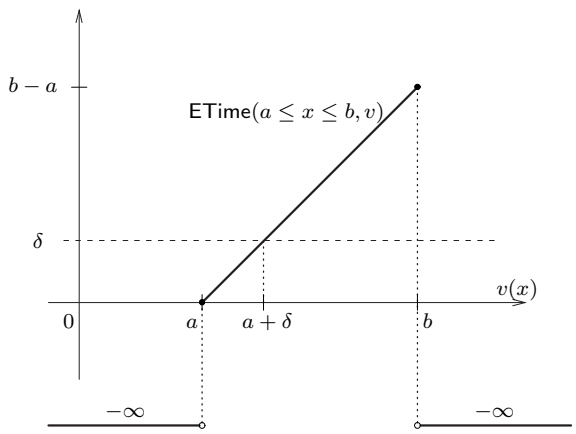


Figure 5.7: We have $\mathsf{ETime}(a \le x \le b, v) > \delta$ iff $v(x) \in]a+\delta, b]$.

Definition 5.11 [Left and right bounds of a rectangular predicate]

Let $\varphi \in \mathsf{Rect}(\mathsf{Var})$ be a rectangular predicate and $x \in \mathsf{Var}$. We define the *left* (resp. *right*) bound $lb(\varphi(x))$ (resp. $rb(\varphi(x))$) of φ for x as follows (recursively):

- if $\varphi \equiv \top$ then $lb(\varphi(x)) = -\infty$ and $rb(\varphi(x)) = +\infty$;
- if $\varphi \equiv \bot$ then $lb(\varphi(x)) = +\infty$ and $rb(\varphi(x)) = -\infty$;
- if $\varphi \equiv x' \ge a$ or $\varphi \equiv x' > a$ then $lb(\varphi(x)) = \begin{cases} a & \text{if } x = x' \\ -\infty & \text{if } x \neq x' \end{cases}$ and $rb(\varphi(x)) = +\infty$;
- if $\varphi \equiv x' \le a$ or $\varphi \equiv x' < a$ then $lb(\varphi(x)) = -\infty$ and $rb(\varphi(x)) = \begin{cases} a & \text{if } x = x' \\ +\infty & \text{if } x \neq x' \end{cases}$
- if $\varphi \equiv x' = a$ then $lb(\varphi(x)) = rb(\varphi(x)) = a$ if $x = x'$ and $lb(\varphi(x)) = -rb(\varphi(x)) = -\infty$ if $x \neq x'$;
- if $\varphi \equiv \varphi_1 \wedge \varphi_2$ then $lb(\varphi(x)) = \max\{lb(\varphi_1(x)), lb(\varphi_2(x))\}$ and $rb(\varphi(x)) = \min\{rb(\varphi_1(x)), rb(\varphi_2(x))\}$

□

In the next lemma, we give the relationship between the condition $\mathsf{ETime}(\varphi, v) \le \delta$ that is used in the rules $(A5.1)$ and $(A5.2)$ of the AASAP semantics (Definition 4.15) and the predicate $_{-\Delta}]\varphi]_0$. We show that the two expressions define the same set of valuations. Observe however that the predicate in Lemma 5.12 is not exactly of the form of the *closed* predicates $\bar{\varphi}_{\mathsf{act}}(\ell)$ and $\bar{\varphi}_{\mathsf{evt}}(\ell, \alpha)$ in Definition 5.5. We come back to that issue after the lemma.

Lemma 5.12 *Let $\varphi \in \mathsf{Rect}(\mathsf{Var})$ be a rectangular predicate and let $\delta \in \mathbb{Q}^{\geq 0}$. For all valuations v over Var, we have:*

$$\mathsf{ETime}(\varphi, v) \leq \delta \text{ iff } v \models_\delta \neg({}_{-\Delta}]\varphi]_0)$$

Proof. Figure 5.7 may be helpful. It shows the value of $\mathsf{ETime}(a \leq x \leq b, v)$ as a function of $v(x)$. Notice that $\mathsf{ETime}(a \leq x \leq b, v) = \mathsf{ETime}(a \leq x < b, v)$. On the other hand, we have $\mathsf{ETime}(a < x \leq b, v) = \mathsf{ETime}(a < x < b, v)$ and those four functions differ only when $v(x) = a$: we have $\mathsf{ETime}(a \leq x \leq b, v) = 0$ and $\mathsf{ETime}(a < x \leq b, v) = -\infty$. However, the caption of Figure 5.7 holds for all $\delta \geq 0$ and all the above predicates.

For each $x \in \mathsf{Var}$, let $a_x = lb(\varphi(x))$ and $b_x = rb(\varphi(x)))$. Then, we have successively:

$\mathsf{ETime}(\varphi, v) \leq \delta$
iff δ is an upper bound of the set $\{t \in \mathbb{R}^{\geq 0} \mid \forall t' \in [0,t] : v - t' \models \varphi\}$
iff $\forall t \in \mathbb{R}^{\geq 0} : (\forall t' \in [0,t] : v - t' \models \varphi) \rightarrow t \leq \delta$
iff $\forall t \in \mathbb{R}^{\geq 0} : (v \models \varphi \wedge v - t \models \varphi) \rightarrow t \leq \delta$
iff $\forall t \in \mathbb{R}^{\geq 0} : v \not\models \varphi \vee v - t \not\models \varphi \vee t \leq \delta$
iff $v \not\models \varphi \vee \forall t > \delta : v - t \not\models \varphi$
iff $(v \models \varphi) \rightarrow \exists x \in \mathsf{Var} : v(x) - \delta \leq a_x$
iff $(v \models \varphi) \rightarrow (\exists x \in \mathsf{Var} : a_x \leq v(x) \leq b_x \ \wedge \ v(x) - \delta \leq a_x)$
iff $(v \not\models \varphi) \vee \exists x \in \mathsf{Var} : a_x \leq v(x) \leq \min\{b_x, a_x + \delta\})$
iff $(\exists x \in \mathsf{Var} : v(x) < a_x \vee v(x) > b_x \vee (\exists x \in \mathsf{Var} : a_x \leq v(x) \leq \min\{b_x, a_x + \delta\})$
iff $\exists x \in \mathsf{Var} : v(x) \leq a_x + \delta \vee v(x) > b_x$
iff $\exists x \in \mathsf{Var} : v(x) \notin (a_x + \delta, b_x]$
iff $v \not\models_\delta {}_{-\Delta}]\varphi]_0$
iff $v \models_\delta \neg({}_{-\Delta}]\varphi]_0)$.

■

The rules $(A5.1)$ and $(A5.2)$ of the AASAP semantics (Definition 4.15) give the condition for the passage of time in the form of a conjunction of disjunctions. The natural encoding of that condition in timed automata is by the use of invariants. Since time can pass in a location as soon as its invariant is satisfied, if the invariant corresponds exactly to the condition of rules $(A5.1)$ and $(A5.2)$, then all is fine. This is why we have the invariant '$d \leq \Delta \vee \bar{\varphi}_{\mathsf{act}}(\ell)$' in each location Out_ℓ ($\ell \in \mathsf{Loc}$) of $\mathcal{E}(A)$, the invariant '$z_\alpha \leq \Delta$' ($\alpha \in \mathsf{Lab_{in}}$) in the location W_1 of each event-watcher W_α (corresponding to the condition $I(\alpha) \leq \Delta$), and the invariant $\bar{\varphi}_{\mathsf{evt}}(\ell, \alpha)$ in the location U_2 of each guard-watcher W^ℓ_α ($\ell \in \mathsf{Loc}$, $\alpha \in \mathsf{Lab_{in}}$).

However, according to Lemma 5.12, the expressions $\bar{\varphi}_{\mathsf{act}}$ and $\bar{\varphi}_{\mathsf{evt}}$ slightly over-approximate the condition $\mathsf{ETime}(\cdot, \cdot) \leq \delta$. Indeed, they correspond to the *closure* of

the exact condition. As a consequence, the construction $\mathcal{F}(A)$ has potentially more behaviours than the expected construction (which should use the exact condition of Lemma 5.12). As we will see in the proof of Theorem 5.10, this is in fact not the case, but it somewhat complicates the proof. Intuitively, this is explained by the fact that the two invariants $I_1 \equiv x \leq \Delta \vee y > 1$ and $I_2 \equiv x \leq \Delta \vee y \geq 1$ are equivalent, in the sense that given any valuations v and v', there exists a timed transition from v to v' under invariant I_1 if and only if there exists such a timed transition under invariant I_2. In Lemma 5.14, we generalize and formalize this argument.

The reason to do this subtle trick is as follows: the encoding $\mathcal{F}(A)$ is intended to be used for verification of the AASAP semantics with automatic verification tools like HyTech or Uppaal. But those tools do not accept *disjunctions* in their input language. This is claimed to be unrestrictive because it is in general easy to build an equivalent Alur-Dill automaton from a timed automaton over multirectangular predicates, by *splitting* components (edges and locations). For edges, the splitting is straightforward, but some care is required for locations.

In a timed automaton over multirectangular predicates, we would split a location ℓ with invariant $\mathsf{Inv}(\ell) \equiv p_1 \vee \cdots \vee p_n$ into n locations ℓ_i with invariants $\mathsf{Inv}(l_i) \equiv p_i$ for $1 \leq i \leq n$. Those locations are connected to each other by silent edges. The initial location and the edges of the automaton are modified as expected. This transformation cannot be done safely (with preservation of emptiness) in general. For example, splitting a location ℓ with invariant $x < 1 \vee x \geq 1$ is not safe: from the location ℓ_1 with invariant $x < 1$, it is impossible to reach the location ℓ_2 with invariant $x \geq 1$ and thus time is blocked and x cannot reach 1. Of course, this is not the case in the original location. A sufficient condition for safe splitting of locations is that the expressions $p_1, \ldots, p_n$ define *closed* sets of valuations. Hence, we use the expressions of Definition 5.5, defining closed sets. The establish the correctness of the construction, we need the following lemma about right-closed predicates.

Definition 5.13 [Right-closed predicates] A parametric multirectangular predicate $\varphi \in \mathsf{MultPRect}(\mathsf{Var})$ is *right-closed* if (i) it does not contain any negation sign '$\neg$'; and (ii) it does not contain any strict inequality sign '$<$'. □

Lemma 5.14 *Given a parametric multirectangular predicate $\varphi \in \mathsf{MultPRect}(\mathsf{Var})$ that is right-closed, let $\bar{\varphi}$ be the predicate obtained by replacing each occurrence of '$>$' by '$\geq$' in φ and let $\breve{\varphi}$ be the predicate obtained by replacing each occurrence of '$\geq$' by '$>$' in φ. For all valuations v over Var, for all $\delta \in \mathbb{Q}^{\geq 0}$ and $T \in \mathbb{R}^{>0}$:*

$$\textit{if } \ \forall t \in]0, T] : v + t \models_\delta \bar{\varphi}, \ \textit{ then } \ \forall t \in]0, T] : v + t \models_\delta \breve{\varphi}.$$

Proof. Without loss of generality, we may assume that φ is in conjunctive normal form and does not contain equality. Since for $\varphi = \varphi_1 \wedge \varphi_2$, we have $\bar{\varphi} = \bar{\varphi_1} \wedge \bar{\varphi_2}$ and $\breve{\varphi} = \breve{\varphi_1} \wedge \breve{\varphi_2}$, it suffices to prove the Lemma when φ is a finite disjunction of constraints

of the form $x_i \bowtie a_i$. For each constraint $p \equiv x \bowtie a$ appearing in φ, let $t_p \in \mathbb{R}$ such that $v(x)+t_p = a_\delta$ where a_δ is equal to respectively $c, c+\delta, c-\delta$ if a is equal to $c, c+\Delta, c-\Delta$. Let P_φ be the set of constraints appearing in φ, and let $T_P = \{t_p \mid p \in P_\varphi\}$.

For an arbitrary $t \in]0, T]$, let us prove that $v + t \models_\delta \breve{\varphi}$. We have $v + t \models_\delta \bar{\varphi}$, and thus $v + t \models_\delta \bar{p}$ for some $p \in P_\varphi$.

- If $t \neq t_p$, then we have trivially $v + t \models_\delta \breve{p}$.
- If $t = t_p$ and $\bar{p} \equiv x \leq a$, then $\bar{p} \equiv \breve{p}$ and we have again $v + t \models_\delta \breve{p}$.
- If $t = t_p$ and $\bar{p} \equiv x \geq a$, then let $t_q = \max(\{t' \in T_P \mid t' < t_p\} \cup \{0\})$. Since $t_p > 0$, we have $t_p > t_q$ and there exists $\hat{t}$ such that $t_q < \hat{t} < t_p$ and $\hat{t} \in]0, T]$. Therefore we have $v + \hat{t} \models_\delta \bar{\varphi}$ and thus $v + \hat{t} \models_\delta \bar{p}'$ for some $p' \in P_\varphi$. Let $p' \equiv x' \bowtie a'$. Either $\bowtie \in \{>, \geq\}$ and then $v + t_p \models_\delta \breve{p}'$ because $t_p > \hat{t}$, or $\bowtie \equiv \leq$ and then since $v(x') + \hat{t} \leq a'$ we have $t_{p'} \geq \hat{t}$ and thus $t_{p'} \geq t_p$. Hence, we also have $v + t_p \models_\delta \breve{p}'$ as $\bar{p}' \equiv \breve{p}'$.

In summary, we have shown that at least one predicate in $\breve{\varphi}$ is satisfied by $v + t$. We conclude that $v + t \models_\delta \breve{\varphi}$.

■

Now, we present the proof of Theorem 5.10 that establishes the mutual simulation between $[\![A]\!]_\delta^{\mathcal{F}}$ and $[\![A]\!]_\delta^{\mathsf{AAsap}}$.

Proof of Theorem 5.10.

In this proof, we use the following notations: for a location $\ell = (\ell_1, \ldots, \ell_n)$ of a synchronized product $A_1 \times \cdots \times A_n$ of timed automata, we define $\ell(A_i) = \ell_i$ for $1 \leq i \leq n$. Let $\hat{\ell} \in \mathsf{Loc}^i$ be a location of A_i, we define $\ell[A_i := \hat{\ell}]$ to be the location ℓ' such that $\ell'(A_i) = \hat{\ell}$ and $\ell'(A_j) = \ell_j$ for $j \neq i$.

Let $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ structured by $(\mathsf{Lab}_{\mathsf{in}}, \mathsf{Lab}_{\mathsf{out}}, \mathsf{Lab}_\tau)$. Let $[\![A]\!]_\delta^{\mathsf{AAsap}}$ be given by the TTS $\mathcal{T}_1 = \langle Q^1, Q_0^1, Q_f^1, \Sigma^1, \rightarrow^1 \rangle$ structured by $(\Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau)$ and $[\![A]\!]_\delta^{\mathcal{F}}$ be given by the TTS $\mathcal{T}_2 = \langle Q^2, Q_0^2, Q_f^2, \Sigma^2, \rightarrow^2 \rangle$ structured by $(\Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau)$. Then, we have $\Sigma^1_{\mathsf{in}} = \Sigma^2_{\mathsf{in}} = \mathsf{Lab}_{\mathsf{in}}$, $\Sigma^1_{\mathsf{out}} = \Sigma^2_{\mathsf{out}} = \mathsf{Lab}_{\mathsf{out}}$ and $\Sigma^1_\tau = \Sigma^2_\tau = \mathsf{Lab}_\tau \cup \overline{\mathsf{Lab}_{\mathsf{in}}} \cup \{\tau\}$.

First, we show that $[\![A]\!]_\delta^{\mathcal{F}} \succeq_{weak} [\![A]\!]_\delta^{\mathsf{AAsap}}$. Let $\mathcal{R} \subseteq Q^2 \times Q^1$ be the set of pairs $((\ell_2, v_2), (\ell_1, v_1, I_1, d_1))$ such that:

1. $\ell_2(\mathcal{E}(A)) = \begin{cases} In_{\ell_1} & \text{if } d_1 \leq \delta \\ Out_{\ell_1} & \text{otherwise} \end{cases}$
2. For all $\alpha \in \mathsf{Lab}_{\mathsf{in}}$: $\ell_2(W_\alpha) = \begin{cases} W_0 & \text{if } I_1(\alpha) = \bot \\ W_1 & \text{if } I_1(\alpha) \neq \bot \text{ and } I_1(\alpha) \leq \delta \\ W_2 & \text{otherwise} \end{cases}$

3. For all $\alpha \in \mathsf{Lab_{in}}$, $\ell \in \mathsf{Loc}$: $\ell_2(W_\alpha^\ell) = \begin{cases} U_1 & \text{if } \ell = \ell_1 \\ U_0 & \text{otherwise} \end{cases}$

4. $\forall x \in \mathsf{Var} : v_2(x) = v_1(x)$;

5. $v_2(d) = d_1$;

6. For every $\alpha \in \mathsf{Lab_{in}}$, if $I_1(\alpha) \neq \bot$ then $v_2(z_\alpha) = I_1(\alpha)$.

Let us show that $\mathcal{R}$ is a weak simulation relation, in the sense of Definition 2.5.

First, we have $Q_f^1 = \varnothing$, and thus trivially $\forall q_1 \in Q_f^1 \cdot \forall q_2 \in Q^2$: if $(q_2, q_1) \in \mathcal{R}$ then $q_2 \in Q_f^2$.

Second, for each $(\ell_1, v_1, I_\bot, 0) \in Q_0^1$ there exists $(\ell_2, v_2) \in Q_0^2$ such that $(q_2, q_1) \in \mathcal{R}$. Indeed since $\mathsf{Init}(\ell)$ is satisfiable for at most one location $\ell \in \mathsf{Loc}$, we can take ℓ_2 such that $\ell_2(\mathcal{E}(A)) = In_{\ell_1}$, for each $\alpha \in \mathsf{Lab_{in}} : \ell_2(W_\alpha) = W_0$ and for any $\alpha \in \mathsf{Lab_{in}}$ and $\ell \in \mathsf{Loc} : \ell_2(W_\alpha^\ell) = U_1$ if $\ell = \ell_1$ and $\ell_2(W_\alpha^\ell) = U_0$ otherwise. And we take v_2 such that $v_2(d) = 0$ and $v_2(x) = v_1(x)$ for all $x \in \mathsf{Var}$.

Third, if $(q_2, q_1) = ((\ell_2, v_2), (\ell_1, v_1, I_1, d_1)) \in \mathcal{R}$ and $(q_1, \sigma, q_1') \in \twoheadrightarrow^1$ for some $\sigma \in \Sigma^1 \setminus \{\tau\}$ (with $q_1' = (\ell_1', v_1', I_1', d_1')$), then we must show that there exists a state $q_2' \in Q^2$ such that $(q_2, \sigma, q_2') \in \twoheadrightarrow^2$ and $(q_2', q_1') \in \mathcal{R}$:

- If $\sigma \in \Sigma_{\mathsf{in}}^1$.

  By rule $(A2)$ of the AASAP semantics (Definition 4.15), since $(q_1, \sigma, q_1') \in \twoheadrightarrow^1$, we have

  - either $I_1(\sigma) = \bot$ and then $q_1' = (\ell_1, v_1, I_1[\sigma := 0], d_1)$. Then, by definition of $\mathcal{R}$ and since $(q_2, q_1) \in \mathcal{R}$, we have $\ell_2(W_\sigma) = W_0$. Thus, there exists a transition $(q_2, \sigma, q_2') \in \twoheadrightarrow^2$ with $q_2' = (\ell_2', v_2')$ where $\ell_2' = \ell_2[W_\sigma := W_1]$ and $v_2' = v_2[z_\sigma := 0]$. This is because W_σ is the only automaton in $\mathcal{F}(A)$ that synchronizes on σ, and it has an edge $(W_0, W_1, \top, \sigma, \{z_\sigma\})$ whose guard is trivially satisfied by v_2 and z_σ is reset so that the invariant $z_\alpha \leq \Delta$ is trivially satisfied by v_2'.

    Finally, it is easy to see that $(q_2', q_1') \in \mathcal{R}$ (in particular, $v_2'(z_\sigma) = I_1'(\sigma) = 0$).
  - or $I_1(\sigma) \neq \bot$ and then $q_1' = q_1$. Then, by definition of $\mathcal{R}$, since $(q_2, q_1) \in \mathcal{R}$ we have either $\ell_2(W_\sigma) = W_1$ or $\ell_2(W_\sigma) = W_2$. In both cases, there exists a transition $(q_2, \sigma, q_2') \in \twoheadrightarrow^2$ with $q_2' = q_2$ since W_σ is the only automaton in $\mathcal{F}(A)$ that synchronizes on σ, and it has two self-loops $(W_1, W_1, \top, \sigma, \varnothing)$ and $(W_2, W_2, \top, \sigma, \varnothing)$ with trivial guard and empty reset.

    Trivially, we have $(q_2', q_1') \in \mathcal{R}$.

- If $\sigma \in \Sigma_{\mathsf{out}}^1$.

By rule $(A1)$ of the AASAP semantics (Definition 4.15), since $(q_1, \sigma, q_1') \in \twoheadrightarrow^1$, we know that there exists an edge $(\ell_1, \ell_1', g, \sigma, R) \in \mathsf{Edg}$ such that $v_1 \models_\delta {}_\Delta[g]_\Delta$ and $v_1' = v_1[R := 0]$. Also $I_1' = I_1$ and $d_1' = 0$.

Since $(q_2, q_1) \in \mathcal{R}$, we have either $\ell_2(\mathcal{E}(A)) = In_{\ell_1}$ or $\ell_2(\mathcal{E}(A)) = Out_{\ell_1}$. If $\ell_2(\mathcal{E}(A)) = In_{\ell_1}$, then we use the edge $(In_{\ell_1}, Out_{\ell_1}, \top, \tau, \varnothing)$ of $\mathcal{E}(A)$ to reach Out_{ℓ_1} (because the invariant of Out_{ℓ_1} contains the invariant of In_{ℓ_1} disjunctively). From location Out_{ℓ_1}, we take the edge $(Out_{\ell_1}, PostOut_{\ell_1,\ell_1'}, {}_\Delta[g]_\Delta, \sigma, R \cup \{d\})$ because the guard is satisfied by v_2 (it is satisfied by v_1, and v_1 agrees with v_2 on the variables in Var), and d is reset so that the invariant of $PostOut_{\ell_1,\ell_1'}$ is satisfied.

Then we can go to $PreIn_{\ell_1'}$ (invariant $d = 0$) and finally to $In_{\ell_1'}$ (invariant $d \leq \Delta$) by synchronizing on out_{ℓ_1} and $in_{\ell_1'}$ respectively. This makes the location of the guard-watchers $W_\alpha^{\ell_1}$ and $W_\alpha^{\ell_1'}$ (for each $\alpha \in \mathsf{Lab_{in}}$) change to U_0 and U_1 respectively. Let $q_2' = (\ell_2', v_2')$ be the new current state in $[\![A]\!]_\delta^{\mathcal{F}}$. It is easy to see that $(q_2', q_1') \in \mathcal{R}$ (in particular, the location of the watchers are correctly updated, we have $v_2'(d) = d_1' = 0$ and for every $x \in R : v_2'(x) = v_1'(x) = 0$).

- If $\sigma \in \Sigma_\tau^1$. If $\sigma \in \mathsf{Lab}_\tau$, the proof is similar to the previous case. Otherwise, we have $\sigma \in \overline{\mathsf{Lab_{in}}}$. Let $\sigma = \bar{\alpha}$ for $\alpha \in \mathsf{Lab_{in}}$.

  By rule $(A3)$ of the AASAP semantics (Definition 4.15), since $(q_1, \bar{\alpha}, q_1') \in \twoheadrightarrow^1$, we know that there exists an edge $(\ell_1, \ell_1', g, \alpha, R) \in \mathsf{Edg}$ such that $v_1 \models_\delta {}_\Delta[g]_\Delta$ and $v_1' = v_1[R := 0]$. Also $I_1(\alpha) \neq \bot$, $I_1' = I_1[\alpha := \bot]$ and $d_1' = 0$.

  We show below that there exists a transition $(q_2, \sigma, q_2') \in \twoheadrightarrow^2$ such that $(q_2', q_1') \in \mathcal{R}$. To do this, we consider the sequence of labels $\bar{\alpha}, out_{\ell_1}, in_{\ell_1'}$ in each component of $\mathcal{F}(A)$ that synchronizes on at least one of those labels.

  1. Event-watchers do not synchronize on out_{ℓ_1} and $in_{\ell_1'}$. Only W_α synchronizes on $\bar{\alpha}$. Since $(q_2, q_1) \in \mathcal{R}$ and $I_1(\alpha) \neq \bot$, we have either $\ell_2(W_\alpha) = W_1$ or $\ell_2(W_\alpha) = W_2$. In the first case, we go to W_2 (that has a trivial invariant) by the edge $(W_1, W_2, \top, \tau, \varnothing)$ of W_α. From W_2, we take the edge $(W_2, W_0, \top, \bar{\alpha}, \varnothing)$, which is enabled since its guard and the invariant of the target location W_0 is trivial.
  2. Guard-watchers W_α^ℓ. If $\ell \neq \ell_1$, then $\ell_2(W_\alpha^\ell) = U_0$ and we use the self-loop $(U_0, U_0, \top, \bar{\alpha}, \varnothing)$. If $\ell = \ell_1$, then $\ell_2(W_\alpha^\ell) = U_1$ and the edge $(U_1, U_3, \top, \bar{\alpha}, \{u_\alpha^{\ell_1}\})$ is used, with reset of $u_\alpha^{\ell_1}$ to reach U_3 (invariant $u_\alpha^{\ell_1} = 0$).

     Now, for each $\beta \in \mathsf{Lab_{in}}$, the guard-watchers $W_\beta^{\ell_1}$ (which synchronizes on out_{ℓ_1}) and $W_\beta^{\ell_1'}$ (which synchronizes on $in_{\ell_1'}$) have an enabled transition labelled by out_{ℓ_1} and $in_{\ell_1'}$ respectively, leading to their location U_0 and U_1 respectively.
  3. The automaton $\mathcal{E}(A)$. Since $(q_2, q_1) \in \mathcal{R}$, we have either $\ell_2(\mathcal{E}(A)) = In_{\ell_1}$ or $\ell_2(\mathcal{E}(A)) = Out_{\ell_1}$. In the first case, we use the edge $(In_{\ell_1}, Out_{\ell_1}, \top, \tau, \varnothing)$ of

$\mathcal{E}(A)$ to reach Out_{ℓ_1} (because the invariant of Out_{ℓ_1} contains the invariant of In_{ℓ_1} disjunctively).

From Out_{ℓ_1}, we take the edge $(Out_{\ell_1}, PostOut_{\ell_1,\ell'_1}, {}_\Delta[g]_\Delta, \bar{\alpha}, R \cup \{d\})$. This is allowed because the guard is satisfied by v_2 (it is satisfied by v_1, and v_1 agrees with v_2 on the variables in Var), and d is reset so that the invariant of $PostOut_{\ell_1,\ell'_1}$ is satisfied.

Then we can go to $PreIn_{\ell'_1}$ (invariant $d = 0$) and finally to $In_{\ell'_1}$ (invariant $d \leq \Delta$) by synchronizing on out_{ℓ_1} and $in_{\ell'_1}$ respectively.

Let $q'_2 = (\ell'_2, v'_2)$ be the new current state in $[\![A]\!]^{\mathcal{F}}_\delta$. It is easy to see that $(q'_2, q'_1) \in \mathcal{R}$ (in particular, the location of the watchers are correctly updated, we have $v'_2(d) = d'_1 = 0$ and for every $x \in R : v'_2(x) = v'_1(x) = 0$).

- If $\sigma = t \in \mathbb{R}^{\geq 0}$.

  By rules $(A5.1)$ and $(A5.2)$ of the AASAP semantics (Definition 4.15), since $(q_1, t, q'_1) \in \twoheadrightarrow^1$, we have:

  $(B1)$ for all edges $(\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab_{out}} \cup \mathsf{Lab}_\tau$,we have:

  $$\forall t' \in [0, t] : d + t' \leq \delta \vee \mathsf{ETime}(g, v + t') \leq \delta$$

  $(B2)$ and for all edges $(\ell, \ell', g, \alpha, R) \in \mathsf{Edg}$ with $\alpha \in \mathsf{Lab_{in}}$, we have:

  $$\forall t' \in [0, t] : d + t' \leq \delta \vee \mathsf{ETime}(g, v + t') \leq \delta \vee (I + t')(\alpha) \leq \delta$$

  By definition of $\mathcal{R}$, since $(q_2, q_1) \in \mathcal{R}$, we have:

  $(C1)$ $v_2(d) = d_1$;

  $(C2)$ $\forall \alpha \in \mathsf{Lab_{in}}$: if $I_1(\alpha) \neq \bot$, then $v_2(z_\alpha) = I_1(\alpha)$;

  $(C3)$ $\forall g \in \mathsf{Rect_c}(\mathsf{Var}) \cdot \forall t' \in \mathbb{R}^{\geq 0}$: $\mathsf{ETime}(g, v_1 + t') \leq \delta$ iff $v_2 + t' \models_\delta \neg({}_{-\Delta}[g]_0)$.

  Proposition $(C3)$ is established by Lemma 5.12 since v_1 and v_2 agree on all the variables in Var.

  We must show that $\mathcal{F}(A)$ can let pass t time units and reach a state q'_2. For this, we show that (i) each automaton in $\mathcal{F}(A)$ can let time t pass and that (ii) the synchronized product of those automata can avoid staying in urgent locations.

  (i) Let us consider each automaton in $\mathcal{F}(A)$:

  - Event-Watchers W_α, for $\alpha \in \mathsf{Lab_{in}}$. If $\ell_2(W_\alpha) = W_0$ or $\ell_2(W_\alpha) = W_2$, then time can pass since the invariant is trivial. Otherwise, we have $\ell_2(W_\alpha) = W_1$. If $t \leq \delta - v_2(z_\alpha)$, then we can stay W_1 for t time units. Otherwise, we wait $\delta - v_2(z_\alpha)$ time units in W_1 (so that the value of z_α is δ) and we jump to W_2 where the invariant is trivial, using the edge $(W_1, W_2, \top, \tau, \varnothing)$.

- Guard-Watchers W^{ℓ}_{α}, for $\ell \in \mathsf{Loc}$ and $\alpha \in \mathsf{Lab_{in}}$. Since $(q_2, q_1) \in \mathcal{R}$, we have either $\ell_2(W^{\ell}_{\alpha}) = U_0$ or $\ell_2(W^{\ell}_{\alpha}) = U_1$. In both cases, time can pass since the invariant is trivial.
- $\mathcal{E}(A)$. Since $(q_2, q_1) \in \mathcal{R}$, we have either $\ell_2(\mathcal{E}(A)) = In_{\ell_1}$ or $\ell_2(\mathcal{E}(A)) = Out_{\ell_1}$. In the first case, either $t \leq \delta - v_2(d)$ and we can stay waiting in In_{ℓ_1}, or $t > \delta - v_2(d)$ and we wait $\delta - v_2(d)$ time units in In_{ℓ_1} followed by a silent jump to Out_{ℓ_1}. We claim that in Out_{ℓ_1}, the remaining $t - (\delta - v^2_1(d))$ time units can pass. In the second case, we claim that t time units can pass in Out_{ℓ_1}.

  Replacing $(C1)$ and $(C3)$ in $(B1)$, we obtain that $\forall g \in G_{\mathsf{act}}(\ell_1) \cdot \forall t' \in [0, t] : (v_2(d) + t' \leq \delta \vee v_2 + t' \models_{\delta} \neg({}_{-\Delta}]g]_0))$. This implies that $v_2 + t' \models_{\delta} d \leq \Delta \vee \bar{\varphi}_{\mathsf{act}}(\ell_1)$, that is $v_2 + t'$ satisfies the invariant of Out_{ℓ_1} for every $0 \leq t' \leq t$.

(ii) It remains to check that we do not need to let time pass in an urgent location. The urgent locations we could enter by letting time pass in $\mathcal{F}(A)$ are the locations ℓ_α (for each $\alpha \in \mathsf{Lab_{in}}$) such that $\ell_\alpha(\mathcal{E}(A)) = Out_{\ell_1}$, $\ell_\alpha(W_\alpha) = W_2$ and $\ell_\alpha(W^{\ell_1}_{\alpha}) = U_1$, where a transition labelled by $\bar{\alpha}$ is proposed.

Assume that after $t' < t$ time units, we get into such an urgent location ℓ_α (for some $\alpha \in \mathsf{Lab_{in}}$). Then, we necessarily have $v_2(d) + t' \geq \delta$ and $v_2(z_\alpha) + t' \geq \delta$ (and $I_1(\alpha) \neq \bot$ since $\ell_\alpha(W_\alpha) = W_2$ and thus $\ell_1(W_\alpha) \neq W_0$). From $(C1)$ and $(C2)$ for every $t'' > t'$ we have $d_1 + t'' > \delta$ and $(I_1 + t'')(\alpha) > \delta$. Therefore, $(B2)$ reduces to $\forall g \in G_{\mathsf{evt}}(\ell_1, \alpha) \forall t'' \cdot t' < t'' \leq t : \mathsf{ETime}(g, v_1 + t'') \leq \delta$, which entails by $(C3)$ that $v_2 + t'' \models_{\delta} \neg({}_{-\Delta}]g]_0)$ and finally $v_2 + t'' \models_{\delta} \bar{\varphi}_{\mathsf{evt}}(\ell_1, \alpha)$ for $t' < t'' \leq t$. Since $\bar{\varphi}_{\mathsf{evt}}(\ell_1, \alpha)$ contains only loose inequalities, we also have $v_2 + t' \models_{\delta} \bar{\varphi}_{\mathsf{evt}}(\ell_1, \alpha)$.

Hence, the invariant of U_2 in $W^{\ell_1}_{\alpha}$ is satisfied by $v_2 + t''$ for all $t'' \in [t', t]$ and so we can jump from U_1 to U_2 and stay there for the remaining $t - t'$ time units.

This trick can be repeated for all urgent locations that we encounter when simulating the timed transition. Since there is a finite number of such locations (at most $|\mathsf{Lab_{in}}|$), there is a *finite number* of silent jumps to a location U_2 of a guard-watcher. Therefore, we have $(q_2, t, q'_2) \in \twoheadrightarrow^2$.

The reader can easily check that we have $(q'_2, q'_1) \in \mathcal{R}$.

Now, we proceed with the second part of the proof showing that $[\![A]\!]^{\mathsf{AAsap}}_{\delta} \succeq_{weak} [\![A]\!]^{\mathcal{F}}_{\delta}$. Let $\mathcal{R}' \subseteq Q^1 \times Q^2$ be the set of pairs $((\ell_1, v_1, I_1, d_1), (\ell_2, v_2))$ such that:

1. $\ell_2(\mathcal{E}(A)) \in \{In_{\ell_1}, Out_{\ell_1}\}$;
2. For all $\ell \in \mathsf{Loc}$, we have $\ell_2(W^{\ell}_{\alpha}) = U_0$ if and only if $\ell \neq \ell_1$;
3. For all $\alpha \in \mathsf{Lab_{in}}$: $I_1(\alpha) = \begin{cases} \bot & \text{if } \ell_2(W_\alpha) = W_0 \\ v_2(z_\alpha) & \text{otherwise} \end{cases}$

4. $\forall x \in \mathsf{Var} : v_1(x) = v_2(x)$;

5. $d_1 = v_2(d)$.

Let us show that $\mathcal{R}'$ is a weak simulation relation, in the sense of Definition 2.5. Notice that $\mathcal{R}' \neq \mathcal{R}^{-}1$ and thus we establish mutual weak simulation and not weak bisimulation.

First, we have $Q_f^2 = \varnothing$, and thus trivially $\forall q_2 \in Q_f^2 \cdot \forall q_1 \in Q^1$: if $(q_1, q_2) \in \mathcal{R}'$ then $q_1 \in Q_f^1$.

Second, for each $(\ell_2, v_2) \in Q_0^2$ it is easy to check that there exists $(\ell_1, v_1, I_\perp, 0) \in Q_0^1$ such that $(q_1, q_2) \in \mathcal{R}'$.

Third, if $(q_1, q_2) = ((\ell_1, v_1, I_1, d_1), (\ell_2, v_2)) \in \mathcal{R}'$ and $(q_2, \sigma, q_2') \in \twoheadrightarrow^2$ for some $\sigma \in \Sigma^2 \setminus \{\tau\}$ (with $q_2' = (\ell_2', v_2')$), then we must show that there exists a state $q_1' \in Q^1$ such that $(q_1, \sigma, q_1') \in \twoheadrightarrow^1$ and $(q_1', q_2') \in \mathcal{R}'$. In this part of the proof, however there are several states $q_2' \in Q^2$ such that $(q_2, \sigma, q_2') \in \twoheadrightarrow^2$, we often assume for the sake of simplicity and *without loss of generality* that q_2' is a particular chosen state. We informally justify that this is not restrictive. First, notice that the assumption that we have made before Definition 2.5 holds trivially since $\mathcal{F}(A)$ has no final states. Therefore, we may disregard the states q_2'' such that $(q_2, \sigma, q_2'') \in \twoheadrightarrow^2$ and $q_2' \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_2''$. Second, we must ensure that all the successors (according to $\twoheadrightarrow^2$) of the other states q_2'' such that $(q_2, \sigma, q_2'') \in \twoheadrightarrow^2$ are also successors of q_2' (with the same label). The reader can check that this is true.

More succinctly, this part of the proof makes the implicit assumption that the state space of $[\![A]\!]_\delta^{\mathcal{F}}$ is restricted to the set $\{q_2 \in Q^2 \mid \exists q_1 \in Q_1 : (q_1, q_2) \in \mathcal{R}'\}$.

- If $\sigma \in \Sigma_{\mathsf{in}}^2$.

  Since the event-watcher W_σ is the only automaton in $\mathcal{F}(A)$ that synchronizes on σ, and $(q_2, \sigma, q_2') \in \twoheadrightarrow^2$ we have:

  - either $\ell_2(W_\sigma) = W_0$ and then assume wlog. that $\ell_2' = \ell_2[W_\sigma := W_1]$. Then, we have $v_2' = v_2[z_\sigma := 0]$ and since $(q_1, q_2) \in \mathcal{R}'$, $I_1(\sigma) = \perp$.

    Therefore, there is a transition $(q_1, \sigma, q_1') \in \rightarrow^1$ with $q_1' = (\ell_1, v_1, I_1[\sigma := 0], d_1)$ (by rule $(A2)$ of the AASAP semantics, Definition 4.15). Hence, we have $(q_1', q_2') \in \mathcal{R}'$ (in particular, $I_1'(\sigma) = v_2'(z_\sigma) = 0$).
  - or $\ell_2(W_\sigma) \neq W_0$ and then assume wlog. that $\ell_2' = \ell_2$ (and thus $q_2' = q_2$). Since $(q_1, q_2) \in \mathcal{R}'$, we have $I_1(\sigma) \neq \perp$, which implies that $(q_1, \sigma, q_1') \in \rightarrow^1$ for $q_1' = q_1$ (by rule $(A2)$ of the AASAP semantics, Definition 4.15). It is then immediate that $(q_1', q_2') \in \mathcal{R}'$.

- If $\sigma \in \Sigma_{\mathsf{out}}^2$.

In this case, $\mathcal{E}(A)$ is the only automaton synchronizing on σ and there must exist an edge $(\ell_1, \ell_1', g, \sigma, R) \in \mathsf{Edg}$ such that $v_2 \models_\delta {}_\Delta[g]_\Delta$ and $v_2' = v_2[R \cup \{d\} := 0]$ (because silent edges labelled by τ do not modify the variables). Also, assume wlog. that $\ell_2'(\mathcal{E}(A)) = In_{\ell_1'}$ (thus the guard-watchers $W_\alpha^{\ell_1}$ and $W_\alpha^{\ell_1'}$ for each $\alpha \in \mathsf{Lab_{in}}$ have moved to U_0 and U_1 respectively, through internal synchronizations on out_{ℓ_1} and $in_{\ell_1'}$ respectively).

Since $(q_1, q_2) \in \mathcal{R}'$, v_1 and v_2 agree on the variables of g and thus $v_1 \models_\delta {}_\Delta[g]_\Delta$. Then by rule $(A1)$ of the AASAP semantics (Definition 4.15), there is a transition $(q_1, \sigma, q_1') \in \rightarrow^1$ where $q_1' = (\ell_1', v_1', I_1, 0)$ and $v_1' = v_1[R := 0]$.

It is easy to check that $(q_1', q_2') \in \mathcal{R}'$ (in particular, $d_1' = v_2'(d) = 0$ and for every $x \in R : v_1'(x) = v_2'(x) = 0$).

- If $\sigma \in \Sigma_\tau^2$. If $\sigma \in \mathsf{Lab}_\tau$, the proof is similar to the previous case. Otherwise, we have $\sigma \in \overline{\mathsf{Lab_{in}}}$. Let $\sigma = \bar{\alpha}$ for $\alpha \in \mathsf{Lab_{in}}$.

  We consider each component of $\mathcal{F}(A)$ and gather the following observations:

  1. By definition of $\mathcal{E}(A)$, there must exist an edge $(\ell_1, \ell_1', g, \alpha, R) \in \mathsf{Edg}$ such that $v_2 \models_\delta {}_\Delta[g]_\Delta$ and $v_2' = v_2[R \cup \{d\} := 0]$. As in the previous case, assume wlog. that $\ell_2'(\mathcal{E}(A)) = In_{\ell_1'}$, thus two silent transitions synchronizing internally on out_{ℓ_1} and $in_{\ell_1'}$ respectively have been taken.
  2. (a) In guard-watchers W_α^ℓ (for $\ell \in \mathsf{Loc}$), the sequence of labels $\bar{\alpha}, out_{\ell_1}, in_{\ell_1'}$ has the following effect: if $\ell \neq \ell_1$ and $\ell \neq \ell_1'$, then W_α^ℓ does not synchronize on out_{ℓ_1} and $in_{\ell_1'}$ and thus $\ell_2(W_\alpha^\ell) = \ell_2'(W_\alpha^\ell) = U_0$ (there is a self-loop on U_0 labelled by $\bar{\alpha}$); if[1] $\ell = \ell_1$, then $\ell_2(W_\alpha^\ell) = U_1$ and $\ell_2'(W_\alpha^\ell) = U_0$ (via U_3); and if $\ell = \ell_1'$, then $\ell_2(W_\alpha^\ell) = U_0$ and assume wlog. that $\ell_2'(W_\alpha^\ell) = U_1$.

     (b) In guard-watchers W_β^ℓ (for $\alpha \neq \beta \in \mathsf{Lab_{in}}$), the location is updated to U_0 if $\ell = \ell_1$, to U_1 if $\ell = \ell_1'$ and it is unchanged otherwise.
  3. For event-watchers (which do not synchronize on out_{ℓ_1} nor on $in_{\ell_1'}$), only W_α synchronizes on $\bar{\alpha}$. Hence, we have either $\ell_2(W_\alpha) = W_1$ or $\ell_2(W_\alpha) = W_2$ and $\ell_2'(W_\alpha) = W_0$.

  From the above observations, and since $(q_1, q_2) \in \mathcal{R}'$, we have $v_1 \models_\delta {}_\Delta[g]_\Delta$ and $I_1(\alpha) \neq \bot$. Hence, by rule $(A3)$ of the AASAP semantics (Definition 4.15), there is a transition $(q_1, \sigma, q_1') \in \rightarrow^1$ with $q_1' = (l_1', v_1', I_1', 0)$ where $v_1' = v_1[R := 0]$ and $I_1' = I_1[\alpha := \bot]$.

  Therefore, it is easy to see that $(q_1', q_2') \in \mathcal{R}'$ (in particular, the location of the watchers are correctly updated and we have $d_1' = v_2'(d) = 0$ and for every $x \in R : v_1'(x) = v_2'(x) = 0$).

[1] The case $\ell_1 \neq \ell_1'$ is handled similarly.

- If $\sigma = t \in \mathbb{R}^{\geq 0}$.

  (1) First, assume that $v_2(d) = 0$. Then $\ell_2(\mathcal{E}(A)) = In_{\ell_1}$. If $t \leq \delta$, then the invariant of In_{ℓ_1} is satisfied by $v_2 + t'$ for all $t' \in [0, t]$ and we may assume wlog. that $\ell_2'(\mathcal{E}(A)) = In_{\ell_1}$. Since $(q_1, q_2) \in \mathcal{R}'$, we have $d_1 = v_2(d)$ and thus $d_1 + t' \leq \delta$ for all $t' \in [0, t]$. Then, by rules $(A5.1)$ and $(A5.2)$ of the AASAP semantics (Definition 4.15), there is a transition $(q_1, t, q_1') \in \rightarrow^1$ with $q_1' = (l_1', v_1', I_1', d_1')$ where $\ell_1' = \ell_1$, $v_1' = v_1 + t$, $I_1' = I_1 + t$ and $d_1' = d_1 + t$. The reader can easily check that we have $(q_1', q_2') \in \mathcal{R}'$.

  On the other hand, if $t > \delta$, then a silent jump to Out_{ℓ_1} must have occurred and thus $\ell_2'(\mathcal{E}(A)) = Out_{\ell_1}$.

  Since the invariant of Out_{ℓ_1} contains the invariant of In_{ℓ_1} disjunctively, we have that $v_2 + t'$ satisfies the invariant of Out_{ℓ_1} for all $t' \in [0, t]$, that is $v_2 + t' \models_\delta d \leq \Delta \vee \bar{\varphi}_{\mathsf{act}}(\ell_1)$. Since $v_2 + t' > \delta$ for all $t' \in (\delta, t]$, we have $v_2 + t' \models_\delta \bar{\varphi}_{\mathsf{act}}(\ell_1)$ for $t' \in (\delta, t]$. By Lemma 5.14 (with $v = v_2 + \delta$), we have $v_2 + t' \models_\delta \breve{\varphi}_{\mathsf{act}}(\ell_1)$ for all $t' \in (\delta, t]$, and thus $v_2 + t' \models_\delta d \leq \Delta \vee \breve{\varphi}_{\mathsf{act}}(\ell_1)$ for all $t' \in [0, t]$. Since $(q_1, q_2) \in \mathcal{R}'$, we have $d_1 = v_2(d)$ and $v_1(x) = v_2(x)$ for all $x \in \mathsf{Var}$. Hence, for all $t' \in [0, t]$, we have $d_1 \leq \delta$ or $v_1 \models_\delta \breve{\varphi}_{\mathsf{act}}(\ell_1)$. By Lemma 5.12 and Definition 5.5, this is equivalent to:

$$\forall g \in G_{\mathsf{act}}(\ell_1) \cdot \forall t' \in [0, t] : d_1 + t' \leq \delta \vee \mathsf{ETime}(g, v_1 + t') \leq \delta)$$

  This shows that the condition of the rule $(A5.1)$ of the AASAP semantics (Definition 4.15) is satisfied.

  Now, we establish the condition of rule $(A5.2)$ for each $\alpha \in \mathsf{Lab}_{\mathsf{in}}$:

  - If $\ell_2(W_\alpha) = W_0$. Since $(q_1, q_2) \in \mathcal{R}'$, we have $I_1(\alpha) = \bot$ and trivially $\forall t' \in [0, t] : (I_1 + t')(\alpha) \leq \delta$ which entails the condition of rule $(A5.2)$. Clearly, $\ell_2'(W_\alpha) = W_0$.
  - If $\ell_2(W_\alpha) = W_1$. Since $(q_1, q_2) \in \mathcal{R}'$, we have $I_1(\alpha) = v_2(z_\alpha)$. If $v_2(z_\alpha) + t \leq \delta$, then the invariant of W_1 is satisfied by $v_2 + t'$ for all $t' \in [0, t]$ and we may assume wlog. that $\ell_2'(W_\alpha) = W_1$. Then, we also have $I_1(\alpha) + t' \leq \delta$ for all $t' \in [0, t]$ which entails the condition of rule $(A5.2)$. On the other hand, if $v_2(z_\alpha) + t > \delta$, then $\ell_2'(W_\alpha) = W_2$ (by an intermediate silent jump). Let us decompose the transition $(q_2, t, q_2') \in \twoheadrightarrow^1$ into $(q_2, \delta, q_2'') \in \twoheadrightarrow^1$ and $(q_2'', t - \delta, q_2') \in \twoheadrightarrow^1$, and assume that the second transition starts with a continuous transition (and not a silent transition). After the first transition, the location of $\mathcal{E}(A)$ is Out_{ℓ_1}, the location of W_α is W_2 and the location of $W_\alpha^{\ell_1}$ is either U_1, U_2 or U_3. It cannot obviously be U_3 because time is blocked there. In the second transition (which is a sequence of silent and continuous transitions from $[\![A]\!]_\delta^{\mathcal{F}}$), when time is passing, the location of $W_\alpha^{\ell_1}$ cannot be U_1 as it would correspond to an urgent location in $[\![A]\!]_\delta^{\mathcal{F}}$. When time is passing, the location of $W_\alpha^{\ell_1}$ is then U_2, and the invariant of U_2 is therefore

satisfied by $v_2 + t'$ for all $t' \in (\delta, t]$. By Lemma 5.14, we conclude that $v_2 + t' \models_\delta \breve{\varphi}_{\mathsf{evt}}(\ell_1, \alpha)$ for all $t' \in (\delta, t]$. By a similar argument as for $G_{\mathsf{act}}(\ell_1)$, we can show that:

$$\forall g \in G_{\mathsf{evt}}(\ell_1, \alpha) \cdot \forall t' \in [0, t] : d_1 + t' \leq \delta \vee \mathsf{ETime}(g, v_1 + t') \leq \delta \vee (I_1 + t')(\alpha) \leq \delta$$

which is the condition of rule $(A5.2)$.

Hence, there is a transition $(q_1, t, q_1') \in \rightarrow^1$ and the reader can check that we have $(q_1', q_2') \in \mathcal{R}'$.

(2) Second, if $v_2(d) \neq 0$. Then, since In_{ℓ_1} is entered with d equal to 0, there must have been in the past a continuous transition from a state $q_2'' = (\ell_2'', v_2'') \in Q^2$ with $v_2''(d) = 0$.

Observe that any transition between state q_2'' and q_2 can only have a label either τ or $\alpha \in \mathsf{Lab_{in}}$, and that such transitions do not modify the variables in $\mathsf{Var} \cup \{d\}$. Hence, we must have $(q_2'', t', q_2') \in \rightarrow^2$ for $t' = v_2(d) + t$, and we can reuse the proof for the case $v_2(d) = 0$ to show the existence of a transition $(q_1'', t', q_1') \in \rightarrow^1$ for $q_1'' \in Q^1$ such that $(q_1'', q_2'') \in \mathcal{R}'$. Therefore, since the $\mathsf{TTS}\ [\![A]\!]_\delta^{\mathsf{AAsap}}$ is normalized, we have $(q_1'', t'', \hat{q}_1) \in \rightarrow^1$ and $(\hat{q}_1, t, q_1') \in \rightarrow^1$ for some $\hat{q}_1 = (\ell_1, \hat{v}_1) \in Q^1$ and $\hat{v}_1$ agrees with v_1 on the variables in $\mathsf{Var} \cup \{d\}$. Hence, by rules $(A5.1)$ and $(A5.2)$ of the AASAP semantics (Definition 4.15), there is also a transition $(q_1, t, q_1') \in \rightarrow^1$ with $q_1' = (l_1', v_1', I_1', d_1')$ where $\ell_1' = \ell_1$, $v_1' = v_1 + t$, $I_1' = I_1 + t$ and $d_1' = d_1 + t$. The reader can easily check that we have $(q_1', q_2') \in \mathcal{R}'$.

■

Theorem 5.10 shows that the compositional construction can be used as an equivalent model to verify safety properties of the AASAP semantics. Hence, we have immediately the following theorem.

Theorem 5.15 *Let A be an* Elastic *controller embedded in an* STTS *environment* Env*. For all $\delta \in \mathbb{Q}^{\geq 0}$, A is correct up to δ if and only if $[\![A]\!]_\delta^{\mathcal{F}} \| \mathsf{Env}$ is empty.*

As a consequence, the AASAP semantics can theoretically be verified using existing model-checkers and the code for an implementation of Elastic controllers can be generated automatically with certified correctness. In practice, the verification of the AASAP semantics using the compositional transformation has been implemented for HyTech and Uppaal in [Leg04] and will be detailed in Martin De Wulf's PhD thesis. From an Elastic controller, the tool generates the watchers and the transformed controller in the input language of Uppaal (with the parameter Δ instantiated to a rational value) or HyTech (Δ is left as a parameter). For the code generation, some academic examples have been treated manually, and the methodology has been applied to a realistic example, the Philips Audio Control Protocol [DDR05b].

5.6 Semi-algorithms for constraint synthesis

We give two semi-algorithms for the *synthesis* of correct values for Δ in the AASAP semantics. These procedures perform either a forward or a backward analysis of the compositional construction, with Δ left as a parameter. To do so, we use the classical view of parametric timed automata as hybrid automata (parameters are variables whose first derivative is 0 in every location). We formalize this view hereafter. Therefore, the semi-algorithms can be implemented in HyTech. They construct the (forward or backward) reachable states and thus are not guaranteed to terminate.

For an interval $\phi \subseteq \mathbb{R}$, we use the notation $[\![A]\!]^{\mathcal{F}}_{\phi}$ to denote $[\![\mathcal{F}'(A)]\!]$ where $\mathcal{F}'(A)$ is a linear hybrid automaton, obtained from $\mathcal{F}(A)$ and ϕ by:

1. splitting the locations with disjunctive invariants, to obtain only conjunctive invariants (as we have sketched after Lemma 5.12);
2. considering the parameter Δ as an additional variable of $\mathcal{F}'(A)$;
3. adding the rectangular constraint $l_\phi \leq \Delta \wedge \Delta \leq r_\phi$ conjunctively to each invariant condition $\mathsf{Inv}(\cdot)$, and the constraint $\dot{\Delta} = 0$ conjunctively to each flow condition $\mathsf{Flow}(\cdot)$.

5.6.1 Forward semi-algorithm

The principle of the forward semi-algorithm (sketched as Algorithm 2) is as follows: first, using Algorithm 1 on page 33, we check if the controller is correct for $\delta = 0$, which corresponds to a perfect hardware (line 1). This corresponds to the usual situation in model-checking: we verify the intrinsic correctness of the controller. If the verification fails, we should conclude that the controller is flawed in its logic. Notice that if the test is passed successfully, it means that the reachable states are computable. Now, we take $\delta = 1$, which corresponds to a hardware executing in the same time scale as the environment (line 2). If the system is safe in this situation, we know by Corollary 4.19 that it is safe for all $\delta \leq 1$. Here, the choice to take $\delta = 1$ is arbitrary and may be replaced by $\delta = +\infty$ (and then replace the interval $[0, 1]$ by $\mathbb{R}^{\geq 0}$ in the algorithm). The goal is to take advantage of the fact that the designer has in general an idea of an upper bound for δ for which it is easy to find a hardware that implements the controller (for instance such that $3\Delta_L + 2\Delta_P < \delta$, according to Theorem 4.20). This way, we reduce the *a priori* range of δ and thus we reduce the size of the reachable states to compute. Clearly, in some particular field of applications, the value $\delta = 1$ should be increased to fit with the particular time scale of the environment, for example in low-level networks or electronics.

We continue the description of the forward semi-algorithm. In the while-loop (line 6), we compute the reachable states of the system $[\![A]\!]^{\mathcal{F}}_{\delta} \| \mathsf{Env}$ for $\delta \in \varphi$ where

Algorithm 2: Forward verification of the AASAP semantics using the compositional construction $\mathcal{F}(\cdot)$.

Data : An ELASTIC controller $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ and an STTS environment Env with final states Q_f.

Result : The set $\varphi = \{\delta \in [0,1] \mid [\![A]\!]^{\mathcal{F}}_{\delta} \| \mathsf{Env} \text{ is empty } \}$.

begin

1 **if** VERIFY$([\![A]\!]^{\mathcal{F}}_{\delta=0} \| \mathsf{Env}) =$ UNSAFE **then return** $\varnothing$;
2 **if** VERIFY$([\![A]\!]^{\mathcal{F}}_{\delta=1} \| \mathsf{Env}) =$ SAFE **then return** $[0,1]$;
3 $S_{old} \leftarrow \varnothing$;
4 $S \leftarrow \mathsf{Reach}([\![A]\!]^{\mathcal{F}}_{\delta=0} \| \mathsf{Env})$;
5 $\varphi \leftarrow [0,1[$;
6 **while** $S \neq S_{old}$ **do**
7 $S_{old} \leftarrow S$;
8 $S \leftarrow \mathsf{Post}_{[\![A]\!]^{\mathcal{F}}_{\varphi} \| \mathsf{Env}}(S)$;
9 $\varphi \leftarrow \varphi \cap \{v(\Delta) \mid \forall((\ell, P), q) \in S : v \in P \rightarrow q \notin Q_f\}$;
10 **return** φ ;

end

φ is an interval of values of δ for which we have not been able so far to prove that $[\![A]\!]^{\mathcal{F}}_{\delta} \| \mathsf{Env}$ is not empty (or unsafe). Before entering the loop, the variable S is initialized to $\mathsf{Reach}([\![A]\!]^{\mathcal{F}}_{\delta=0} \| \mathsf{Env})$, which is a set of states that are reachable for any $\delta \geq 0$ (according to Theorem 4.18 and Theorem 5.10, *faster is better*). At each iteration, S is replaced by its one-step successors computed in the TTS of $[\![A]\!]^{\mathcal{F}}_{\delta} \| \mathsf{Env}$ with $\delta \in \varphi$ (line 8). If we find a state in S that is a final state for Env, we know that the corresponding value of the parameter Δ (seen as a variable of a linear hybrid automaton in the algorithm) is unsafe, so we remove it from φ (line 9). Again, by the *faster is better* property, the set φ is necessarily an interval. If the algorithm terminates, we have $S = S_{old}$ and S contains the reachable states of $[\![A]\!]^{\mathcal{F}}_{\phi} \| \mathsf{Env}$. The set φ gives immediately an answer to both the [Existence] and the [Maximization] flavours of the robust safety control problem (Definition 4.17).

Algorithm 2 is not guaranteed to terminate as it uses the semi-algorithm VERIFY which tries to compute the reachable states of a linear hybrid automaton. Notice that the reachable states are in general not "computable" even for the class of Alur-Dill automata for which the emptiness problem is however decidable (Proposition 2.17). The second reason why our algorithm may not terminate is that the set S computed by the loop (lines 6-9) may never stabilize. A simple example of a timed automaton for which this happens is given in Section 6.5. If the guards of the automaton of Figure 6.1 are enlarged by δ (replacing $x \leq 2$ by $x \leq 2 + \delta$, $y \geq 2$ by $y \geq 2 - \delta$, etc.), it can be shown that the point $v = (\ell_2, x = 0, y = 2)$ (and hence the location err) is reachable no matter the value of $\delta > 0$. However, as δ decreases, the number of transitions that are necessary to reach v increases. Therefore, the loop of Algorithm 2 will need roughly

$O(n)$ steps to remove $\delta = \frac{1}{n}$ from the interval ϕ, and so it will not terminate on that example.

The decidability of the [Existence] flavour of the robust safety control problem is established in Chapter 6.

5.6.2 Backward semi-algorithm

The backward semi-algorithm (sketched as Algorithm 3) is the dual of the forward semi-algorithm. We compute in S the states that can reach the final states by iterating the one-step predecessors operator Pre. The interval φ is reduced at each iteration by removing the values of Δ corresponding to an initial state in S.

Algorithm 3: Backward verification of the AASAP semantics using the compositional construction $\mathcal{F}(\cdot)$.

Data : An ELASTIC controller $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ and an STTS environment Env with initial states Q_0.

Result : The set $\varphi = \{\delta \in [0,1] \mid [\![A]\!]^{\mathcal{F}}_{\delta} \| \mathsf{Env} \text{ is empty } \}$.

begin

1 **if** VERIFY$([\![A]\!]^{\mathcal{F}}_{\delta=0} \| \mathsf{Env}) =$ UNSAFE **then return** $\varnothing$;

2 **if** VERIFY$([\![A]\!]^{\mathcal{F}}_{\delta=1} \| \mathsf{Env}) =$ SAFE **then return** $[0,1]$;

3 $S_{old} \leftarrow \varnothing$;

4 $S \leftarrow \mathsf{Reach}^{-1}([\![A]\!]^{\mathcal{F}}_{\delta=0} \| \mathsf{Env})$;

5 $\varphi \leftarrow [0,1[$;

6 **while** $S \neq S_{old}$ **do**

7 $S_{old} \leftarrow S$;

8 $S \leftarrow \mathsf{Pre}_{[\![A]\!]^{\mathcal{F}}_{\varphi} \| \mathsf{Env}}(S)$;

9 $\varphi \leftarrow \varphi \cap \{v(\Delta) \mid \forall((\ell, P), q) \in S : v \in P \cap [\![\mathsf{Init}(\ell)]\!] \rightarrow q \notin Q_0\}$;

10 **return** φ ;

end

Several variations of those two semi-algorithm are possible. First, we can interleave the forward and backward approaches. At each iteration of the while-loop, we compute both a forward step ($\mathsf{Post}(\cdot)$) and a backward step ($\mathsf{Pre}(\cdot)$), and we use the conjunction of the constraints computed in each semi-algorithm. This way, the constraint is stronger and that should facilitate further computation steps by reducing the state space for the parameter. Second, we can compute the reachable states in the forward algorithm (and the states that can reach the final states in the backward algorithm) more efficiently by storing in S only the *newly* reachable states at each iteration. This is a usual trick that we implement with a new variable R, initialized to $R \leftarrow \varnothing$ at the beginning of the algorithms, and the following two lines inserted after line 8:

8.1 $S \leftarrow \mathsf{Weakdiff}(S, R)$;
8.2 $R \leftarrow (R \cup S)$;

At the beginning of the loop, R contains the reachable states computed in the previous iterations, and S is the set of states that have been discovered reachable at the previous iteration. The new value of S is the set of successors of the ancient S, from which the set R is removed. In fact, the operator $\mathsf{Weakdiff}(S, R)$ computes an over-approximation of the difference $S \setminus R$. Computing the exact difference is computationally costly and model-checkers like HyTech offer weaker operators that are much faster to compute. The approximation is conservative in the sense that S still contains the newly reachable states. The choice between the exact operator and an approximated one depends on the specific application, and both should be tried. Third, we can further reduce the state space by adding an invariant to the system: in the forward algorithm, at each iteration we can intersect the reachable states S with the set $\mathsf{Reach}^{-1}([\![A]\!]^{\mathcal{F}}_{\delta=1} \| \mathsf{Env})$. This way, we remove states that cannot reach the bad states of Env for $\delta = 1$ (and thus for any $\delta \leq 1$). A similar invariant for the backward algorithm is $\mathsf{Reach}([\![A]\!]^{\mathcal{F}}_{\delta=1} \| \mathsf{Env})$.

5.7 Conclusion

In Chapter 4, we have presented a new methodology to obtain a real-time controller that is correct and implementable, with the formalism of timed automata. One of the components of the methodology is the program semantics, that provides a realistic model of a real-time implementation: the characteristics of real systems, such as delays in synchronizations, non-zero computation times and finite precision of digital clocks are taken into account. The other component is the AASAP semantics. The correctness of a model (with regard to the AASAP semantics) implies the correctness of the implementation (with regard to the program semantics). The interest of this stronger relation is that the AASAP semantics can be algorithmically verified, as we have shown in Chapter 5. Therefore, we can formally establish that a controller is implementable on a given hardware, with known characteristics.

In Chapter 6, we solve the problem of deciding the *existence* of a hardware such that the controller is implementable. A positive answer to that problem means that a correct implementation can be obtained by construction, if it is executed on a sufficiently fast hardware. On the other hand, a negative answer means that the controller is not correct when the classical and idealized semantics is perturbed by the slightest uncertainty, and thus that it is not robust. A reasonable conclusion is that robustness is a necessary and sufficient condition of implementability.

Chapter 6

Robustness of Timed Automata

> A necessary condition for existence is to be a nonmember of the class of things which do not exist. (...) No one knows whether this condition is sufficient as well.
>
> Anonymous, *Ask-A-Scientist Archive, http://www.newton.dep.anl.gov/.*

6.1 Introduction

In this chapter, we study a theoretical problem that is closely related to the [Existence] flavour of the robust safety control problem (see Definition 4.17). Our main result is that this problem is decidable.

We define a family of semantics for timed automata that is parameterized by two parameters: Δ and ε. We call this semantics the *enlarged semantics*. The parameter Δ has a similar meaning as in the previous chapter. It is used to enlarge the set of valuations that satisfy a guard. The second parameter ε models another type of perturbation that applies to the rate of the clocks. Instead of being perfectly synchronized with the real time (time in the environment), the clocks of the controller may drift by ε. Their derivative lie in the interval $[1-\varepsilon, 1+\varepsilon]$. There are several reasons for introducing this new semantics of timed automata. First, the perturbations induced by Δ and ε apply uniformly to *all* guards and *all* clocks and so the model is easier to understand and to reason about. Furthermore, the perturbations should be applied to both the controller and the environment. Because of that, the enlarged semantics is slightly more permissive than the AASAP semantics (formally it simulates the AASAP semantics) and the decidability result cannot formally be extended to the [Existence] flavour of the robust safety control problem. Even though, we believe it is not reasonable to implement a controller that is correct according to the AASAP semantics but not according to the enlarged semantics, as it would mean that the model is not robust against arbitrarily small perturbations Δ or ε. Second, there is an interesting

paper by Puri about timed automata perturbed only by drifts (so that Δ is always equal to 0) [Pur98]. The new results that we present follow the general ideas of Puri's paper (*e.g.* they are based on a careful study of the structure of limit cycles of timed automata, a fundamental notion introduced by Puri) but we needed new techniques to deal with the imprecisions on guards instead of the drifts on clocks. Also, the proofs in Puri's paper are not always fully convincing, so we had to reprove a large number of his lemmas for establishing our proof, and we had to correct one of them. Third, as Puri sketches it in his paper, there is a tight connection between the two types of perturbations, and therefore it makes sense to mix them in a new semantics. Finally, notice that the results of Chapter 5 can be adapted in the presence of drifts in clocks of the program semantics, as done in [DDR05a].

6.2 Related Works

In Section 4.5, we have given an overview of the previous works in verification about bridging the gap between mathematical models for real-time and their physical counterpart. There were attempts to reduce the expressive power of timed and hybrid automata, to introduce fuzziness, or to define semantics that are *robust* against small perturbations. In this section, we give samples from the literature that illustrate those approaches, and we compare them with the AASAP semantics, focusing on the decidability aspects.

In [GHJ97], a variant of the classical semantics of Alur-Dill automata is studied. In the classical definition, closed and open automata may have dramatically different behaviours. However, a physical realization of the two models could not distinguish open and closed inequalities and more generally could not distinguish arbitrarily close behaviours. The authors define a new acceptance condition for timed automata that is robust with regard to small perturbations like replacing a strict inequality by a loose one. The language of a robust timed automaton is a set of *tubes* (the open sets of a topology defined by some usual metrics) rather than a set of trajectories that reach the final states. A tube is accepted by a timed automaton if the tube contains a dense set of accepted trajectories. With that definition, a trajectory that is accepted by the classical semantics but that is isolated (because for example, one of the guards is an equality constraint) is now rejected. Symmetrically, a rejected trajectory that is not isolated is now accepted. It is shown that the definition is independent of the choice of a "natural" metrics, and that tube acceptance coincides with trajectory acceptance for open timed automata. For the important verification questions, robust timed automata have the same properties as classical timed automata: their emptiness is decidable (it is PSpace-Complete), they are not determinizable, not closed under complement and the problem of universality is undecidable [HR00]. For the slightly more general class of open rectangular automata, emptiness is undecidable. It may be concluded that the inherent precision (*e.g.* through the use of equality constraint) of timed and

hybrid automata is not the deep cause of the undecidability results.

In [Frä99], the author tends to contradict that claim for hybrid systems, with a definition of robustness that looks more similar to the guard enlargement of the AASAP semantics: in the computation of the reachable states, a perturbed version of the post operator is used. Given a *noise level* ϵ, the set that is computed by the classical operator is augmented by all the neighbour states that are at distance at most ϵ. It is argued that a realistic system should be either safe or unsafe both in the classical and perturbed version of the reachability analysis, because noise, no matter how small, is inevitably present in real-world applications. Such systems are called robust. The central result of the paper is that safety is decidable for robust hybrid systems. However, the interest of the result is somewhat limited because it is undecidable to determine whether a given hybrid automaton is robust or not.

Those works of Fränzle and Henzinger *et al.* consider timed and hybrid automata in continuous time, and give decidability results for robust reachability. In that respect, the main result of this chapter is similar but for a different definition of robustness. However, they do not provide a formal argument to show that their definition of robustness leads to models of real-time systems that are implementable. This is done for the AASAP semantics in Chapter 4.

In [AT04], a decidability result is presented for the emptiness problem of *lazy hybrid automata*. This is a class of rectangular automata with perturbed guards in a sampled discrete-time semantics: the length of the timed transitions is equal to one unit of time, but the mode switches can occur in some interval during the timed evolution, thus in a fuzzy (or lazy) way. It contrasts with the usual discrete-time semantics where the discrete transitions must occur at integer instants. The result is interesting because it does not require the rectangular automata to be initialized and that removes an important restriction for the designer. The drawback is the need to choose a fixed granularity for time and a fixed level of fuzziness for guards. In practice, this is perhaps a low price to pay for decidability.

In the context of real-time temporal logics, it has also been observed that fuzziness reduces the expressive power and may lead to decidability. In [AFH96], it is shown that a temporal logics becomes decidable when disallowing equalities in the timing constraints. More recently, *robust* interpretations of real-time logics have been introduced where the timing constraints are slightly relaxed, and the relation with implementability (*i.e.* the transfer of the properties verified by the model to the implementation) has been studied [HVG04, HMP05]. There are certainly many other interesting results in the literature, but the study of temporal logics is out of the scope of this thesis.

Later in this chapter, we discuss in more details the important work by Puri about the dynamical properties of timed automata [Pur98]. An extension of that work to more general dynamical systems has been proposed in [AB01]. Finally, we mention a recent result for timed automata with a single clock: if the clock may drift by $\varepsilon > 0$, then there exists a *deterministic* timed automaton that accepts the same timed language [ATM05].

Therefore, timed automata with a single drifting clock are complementable. This result is of interest for systems that are specified by a product of 1-clock timed automata, a common situation in circuit theory [BS91].

6.3 The implementability problem

We define the new parametric semantics for timed automata. Since we are only interested in safety properties, the definition does not take into account the invariants. As already mentioned after Definition 2.13, this is not surprising since invariants can be transfered to guards while preserving the reachability properties. Further, we assume that the constants are integers and the variables are bounded by the largest constant M of the timed automaton. Accordingly, we call *bounded region automaton* the region automaton restricted to bounded regions. This is not restrictive for modeling purposes because clocks that run over the largest constant can be reset, and the finite structure of the timed automaton can be used to remember the status of each clock (either *exact* or *over* M). For further execution, the guards are evaluated according to that status, which is correct because the truth value of a guard does not depend on the precise value of a clock as soon as it is larger than M.

Definition 6.1 Given an Alur-Dill automaton $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ with n clocks ($\mathsf{Var} = \{x_1, \ldots, x_n\}$) and largest constant M, and given two numbers $\Delta, \varepsilon \in \mathbb{R}^{\geq 0}$, the *enlarged semantics* of A is the TTS $[\![A]\!]^{\varepsilon}_{\Delta} = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$ where :

- $Q = \{(\ell, v) \mid \ell \in \mathsf{Loc} \wedge v : \mathsf{Var} \rightarrow [0, M+1]\}$;
- $Q_0 = \{(\ell, v) \in Q \mid v \models \mathsf{Init}(\ell)\}$;
- $Q_f = \{(\ell, v) \in Q \mid v \models \mathsf{Final}(\ell)\}$;
- $\Sigma = \mathsf{Lab}$;
- The relation $\rightarrow \subseteq Q \times (\Sigma \cup \mathbb{T}) \times Q$ is defined as follows:
  - Discrete transitions. For $\sigma \in \mathsf{Lab}$, $((\ell, v), \sigma, (\ell', v')) \in \rightarrow$ iff there exists an edge $(\ell, \ell', g, \sigma, R) \in \mathsf{Edg}$ such that $v \in \mathcal{N}_{\infty}([\![g]\!], \Delta)$ and $v' = v[R := 0]$.
  - Timed transitions. For each $t \in \mathbb{T}$, $((\ell, v), t, (\ell', v')) \in \rightarrow$ iff $\ell' = \ell$ and $v' - v \in \mathcal{N}_{\infty}(t, \varepsilon t)$, that is $(1-\varepsilon)t \leq v'(x_i) - v(x_i) \leq (1+\varepsilon)t$ for all $1 \leq i \leq n$.

□

In the sequel, we often write $[\![A]\!]^\varepsilon$ to denote $[\![A]\!]^\varepsilon_0$ and $[\![A]\!]_\Delta$ to denote $[\![A]\!]^0_\Delta$ (this overrides the similar notation of Section 5.3). On the other hand, we have $[\![A]\!]^0_0 = [\![A]\!]$.

For Δ and ε fixed, the enlarged semantics can be seen as an initialized rectangular automaton. We have already mentioned that for this class of automata, the emptiness problem is decidable [HKPV98]. However, it is an open question whether it is possible to decide the *existence* of $\Delta > 0$ such that the enlarged semantics is empty. It is the essential result of this chapter to answer positively to that question.

Definition 6.2 Given an Alur-Dill automaton A, the *implementability problem* asks whether there exists $\Delta \in \mathbb{R}^{>0}$ such that $[\![A]\!]^0_\Delta$ is empty. □

Remark Alur-Dill automata allow both strict and non-strict inequalities in predicates. In the presence of guard enlargement, it would not be restrictive to allow only non-strict inequalities. Indeed, consider an Alur-Dill automaton A and the closure automaton $\widehat{A}$ resulting from A by replacing strict inequalities by non-strict inequalities. It appears obviously that:

$$\mathsf{Reach}([\![\widehat{A}]\!]^\varepsilon_{\frac{\Delta}{2}}) \subseteq \mathsf{Reach}([\![A]\!]^\varepsilon_\Delta) \qquad \text{and} \qquad \mathsf{Reach}([\![A]\!]^\varepsilon_\Delta) \subseteq \mathsf{Reach}([\![\widehat{A}]\!]^\varepsilon_\Delta),$$

and hence the implementability problem on A is equivalent to the the implementability problem on $\widehat{A}$. Therefore, in the sequel, we concentrate on closed timed automata. This class has the property that the set of reachable states is closed.

In the rest of the chapter, we make an assumption on the type of cycles in timed automata, in order to prove that the implementability problem is decidable. This assumption is slightly restrictive. We discuss this issue below.

Definition 6.3 A *progress cycle* in the region automaton of a timed automaton is an elementary cycle in which each clock of the automaton is reset at least once. □

Assumption 6.4 We only consider timed automata whose elementary cycles in the region automaton are all progress cycles.

This assumption was made by Puri in [Pur98]. It is weaker than the classical non-Zeno assumptions in the literature. For example in [AMPS98], the authors impose that "in every cycle in the transition graph of the automaton, there is at least one transition which resets a clock variable x_i to zero, and at least one transition which can be taken only if $x_i \geq 1$". Other natural hypotheses would be to ask that every cycle in the region automaton has a timed transition, or that every cycle in the region automaton contains a *time-elapsing* region, that is a region r such that $\exists v \in r \cdot \exists t > 0 : v + t \in r$. Again, Assumption 6.4 is weaker. So, that assumption should not be an obstacle to the design of real-time systems with timed automata, as we only rule out automata that are clearly weird. Many interesting intermediate results in this chapter hold without

that assumption so we will always explicitly refer to it when it is needed to prove a claim.

In the next sections, we establish a strong link between robustness and implementability of timed automata, and we prove that the implementability problem for Alur-Dill automata is decidable under Assumption 6.4. Before that, we need some technical definitions and properties related to the regions of timed automata.

6.4 Properties of Zones and Regions

We review the important properties of the regions of timed automata, for a heavy use in the sequel. The reader is invited to refer to Section 2.4 for the basic definitions related to timed automata.

According to Definition 2.14, a region of a timed automaton contains a set of valuations that agree on the integral part of the clocks and on the ordering of their fractional parts. We make this characterization of regions more concrete by introducing the following representation of regions [Pur98]:

Definition 6.5 Let $\mathbb{N}^{\perp} = \mathbb{N} \cup \{\perp\}$ be the set of $\perp$-*integers*. Given an Alur-Dill automaton A with n clocks ($\mathsf{Var} = \{x_1, \ldots, x_n\}$) and largest constant M, we represent a region of A by:

1. a tuple of $(a_1, \ldots, a_n)$ of $\perp$-integers;
2. and a tuple $(X_0, X_1, \ldots, X_k)$ of $k+1$ $(0 \leq k \leq n)$ sets of clocks that form a partition of the clocks that have a value less than M, except that X_0 can be empty. Formally, let $\mathsf{Var}^{\leq M} = \{x_i \in \mathsf{Var} \mid a_i \neq \perp\}$. We ask that $\mathsf{Var}^{\leq M} = X_0 \cup \cdots \cup X_k$, $X_i \cap X_j = \varnothing$ if $i \neq j$ and $X_i \neq \varnothing$ for all $1 \leq i \leq k$.

The region characterized by a tuple $(a_i)_{1 \leq i \leq n}$ and $(X_i)_{0 \leq i \leq k}$ is the set of all valuations $v : \mathsf{Var} \to \mathbb{R}^{\geq 0}$ such that:

1. For all $x_i \in \mathsf{Var}$: $v(x_i) > M$ iff $a_i = \perp$ and if $a_i \neq \perp$ then $\lfloor v(x_i) \rfloor = a_i$;
2. for all $x, y \in \mathsf{Var}^{\leq M}$: $\langle v(x) \rangle < \langle v(y) \rangle$ iff for some $i < j$, $x \in X_i$ and $y \in X_j$;
3. and for all $x \in \mathsf{Var}^{\leq M}$: $\langle v(x) \rangle = 0$ iff $x \in X_0$.

□

Notice that the size of this representation is bounded because the integers a_i are less than or equal to M. As a consequence, an upper bound of the number of regions can easily be derived: the number $W(M, n)$ of regions is bounded by $(M+2)^n \cdot n! \cdot 2^n$.

A more classical way to represent regions is by the set of constraints it satisfies. Our representation of a region r is easily translated to a set of constraints that are satisfied by (and only by) valuations of r. A valuation v belongs to a region represented by $(a_i)_{1\leq i\leq n}$ and $(X_i)_{0\leq i\leq k}$ if and only if v satisfies the following constraints [AD94]:

- $x_i > M$ for each x_i such that $a_i = \perp$;
- $x_i = a_i$ for each $x_i \in X_0$;
- $a_i < x_i < a_i + 1$ for each $x_i \in X_l$ for some $l > 0$;
- $x_i - a_i < x_j - a_j$ for each x_i, x_j such that $x_i \in X_l$ and $x_j \in X_m$ for some $0 < l < m$;
- $x_i - a_i = x_j - a_j$ for each $x_i, x_j \in X_l$ for some $l > 0$.

Given a region r of a timed automaton, we denote by $[r]$ its topological closure and $[r]$ is called a *closed region*. For a valuation v, we write $[v]$ for the closed region containing v.

Example In a timed automaton with 5 clocks and largest constant $M = 8$, a region r represented by $(1, 3, 5, \perp, 2)$ and $(\{x_1, x_3\}, \{x_2\}, \{x_5\})$ satisfies the following constraints:

$$0 = x_1 - 1 = x_3 - 5 < x_2 - 3 < x_5 - 2 < 1 \wedge x_4 > 8$$

The closure $[r]$ of r then satisfies:

$$0 = x_1 - 1 = x_3 - 5 \leq x_2 - 3 \leq x_5 - 2 \leq 1 \wedge x_4 \geq 8$$

The constraints in the representation of regions are rectangular or diagonal. The sets that can be represented by such constraints are called *zones* in the literature. Zones play an important role in the algorithmic approach of the reachability analysis of timed automata [Bou01]. They are used to represent convex unions of regions[1]. Since we focus on closed timed automata, we use a slightly different definition. In our setting, zones are considered as closed sets.

Definition 6.6 A *zone* $Z \subseteq \mathbb{R}^n$ is a *closed set* defined by inequalities of the form

$$x_i - x_j \leq m_{ij}, \quad \alpha_i \leq x_i, \quad x_i \leq \beta_i$$

where $1 \leq i, j \leq n$ and $m_{ij}, \alpha_i, \beta_i \in \mathbb{Z}$. A set of states is called a *zone-set* if it is a finite union of sets of the form $\{\ell\} \times Z$ where ℓ is a location and Z is a zone. □

[1]However, not all convex unions of regions can be represented by zones

A typical example of a zone-set is the set $\bigcup_{\ell \in \mathsf{Loc}} \{\ell\} \times [\![\mathsf{Final}(\ell)]\!]$ of final states of a closed timed automaton.

Related to the algorithmic analysis of timed automata, an efficient data structure has been introduced to represent zones: the difference bound matrices (DBM) [Dil89]. We briefly introduce DBM and show how the basic operations that are useful for reachability analysis are computed.

Let $x_1, \ldots, x_n$ be the clocks of an Alur-Dill automaton. The idea of the DBM is to represent all constraints uniformly, by diagonal constraints of the form $x_i - x_j \leq a$ with $a \in \mathbb{Z} \cup \{+\infty\}$. For bounded zones, the range of a can be reduced to $\mathbb{Z} \cap [-M, M]$ where M is the largest constant of the timed automaton. The rectangular constraints are written $x_0 - x_i \leq -\alpha_i$ and $x_i - x_0 \leq \beta_i$ where x_0 is a special "variable" whose value is always 0.

A DBM is a $(n+1) \times (n+1)$ matrix $\mathbf{M} = \left(m_{i,j}\right)_{0 \leq i,j \leq n}$ where each $m_{i,j}$ is of the form $(a_{i,j}, \prec_{i,j})$ where $\prec_{i,j} \in \{<, \leq\}$ and $a_{i,j} \in \mathbb{Z}$ is called a *bound*). In the sequel, we only consider DBM that represent closed sets, that is DBM where $\prec_{i,j}$ is always $\leq$. The set of valuations represented by the DBM $\mathbf{M} = \left(m_{i,j}\right)_{0 \leq i,j \leq n}$ is:

$$[\![\mathbf{M}]\!] = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid \forall\, 0 \leq i, j \leq n : x_i - x_j \leq m_{ij} \wedge\ x_0 = 0\}.$$

A DBM is associated to a complete directed graph with nodes $0, 1, \ldots, n$ and edges (i, j) labeled by m_{ij}. In this graph, the *length* of a path is the sum of the labels of the edges in the path. It is easy to see that the length of a path from node i to node j is an upper bound of the difference $x_i - x_j$. The length of the shortest paths between nodes gives the tightest bounds on the variables and difference of variables. This allows to define a *normal form* for DBM that corresponds to the shortest path closure of the associated directed graph. If the graph contains a cycle of negative length, the shortest path closure does not exist and the DBM represents the empty set as a constraint of the form $x_i - x_i \leq m$ with $m < 0$ is unsatisfiable. Hence, for nonempty DBM in normal form, we have $m_{i,i} = 0$ for all $0 \leq i \leq n$.

Operations on sets represented by DBM are executed by syntactic transformations on the DBM. Some of those operations require the normal form. We present the operations that will be useful in the sequel. Other operations like the difference of two DBM and the inclusion test are definable (see [ACD+92, Yov96, CGP99] for details).

- Intersection: given two DBM $\mathbf{M} = \left(m_{i,j}\right)_{0 \leq i,j \leq n}$ and $\mathbf{M}' = \left(m'_{i,j}\right)_{0 \leq i,j \leq n}$, let $\mathbf{M}''$ be the DBM such that $m''_{i,j} = \min\{m_{i,j}, m'_{i,j}\}$. Then, we have $[\![\mathbf{M}'']\!] = [\![\mathbf{M}]\!] \cap [\![\mathbf{M}']\!]$. It is not required that $\mathbf{M}$ and $\mathbf{M}'$ are in normal form. In either case the result is not necessarily in normal form.

- Time passing: given a DBM in normal form $\mathbf{M} = \left(m_{i,j}\right)_{0 \leq i,j \leq n}$, let $\mathbf{M}^{\nearrow}$ be the DBM $\left(m'_{i,j}\right)_{0 \leq i,j \leq n}$ such that $m'_{i,0} = \infty$ and $m'_{i,j} = m_{i,j}$ for all $0 \leq i, j \leq n$ with

$j \neq 0$. This removes the upper bound on all the clocks. We have $[\![\mathbf{M}^{\nearrow}]\!] = \{v+t \mid v \in [\![M]\!] \wedge t \in \mathbb{R}^{\geq 0}\}$ and $\mathbf{M}^{\nearrow}$ is in normal form.

- Reset: given a DBM in normal form $\mathbf{M} = \left(m_{i,j}\right)_{0 \leq i,j \leq n}$ and a clock x, let $\mathbf{M}[x := 0]$ be the DBM $\left(m'_{i,j}\right)_{0 \leq i,j \leq n}$ such that for all $0 \leq i, j \leq n$ with $i \neq j$:

$$m'_{i,j} = \begin{cases} m_{0,j} & \text{if } x = x_i \\ m_{i,0} & \text{if } x = x_j \\ m_{i,j} & \text{otherwise} \end{cases}$$

  We have removed all the bounds involving x and set x to zero. We have $[\![\mathbf{M}[x := 0]]\!] = \{v[x := 0] \mid v \in [\![M]\!]\}$. We define similarly $\mathbf{M}[R := 0]$ for $R \subseteq \{x_1, \ldots, x_n\}$. The result is in normal form.

- Emptiness test: given a DBM $\mathbf{M} = \left(m_{i,j}\right)_{0 \leq i,j \leq n}$, we have $[\![\mathbf{M}]\!] = \varnothing$ if and only if there is a cycle in the directed graph associated to $\mathbf{M}$ whose length is negative. The emptiness test is realized by the shortest path algorithm used to put DBM in normal form.

We continue our review with the notion of *vertex* of a region. For a set $S \subseteq \mathbb{R}^n$, let $\mathsf{Conv}(S)$ be the convex hull of S.

Definition 6.7 Let r be a bounded region of an Alur-Dill automaton. The set of *vertices* of r is the smallest set of points $S(r) = \{v_0, v_1, \ldots, v_k\}$ such that $[r] = \mathsf{Conv}(\{v_0, v_1, \ldots, v_k\})$. □

Definition 6.7 is meaningful because bounded regions are polytopes. The set $S(r)$ is unique and the number of vertices is at most $n+1$, where n is the number of clocks of the automaton.

Lemma 6.8 *The vertices of a bounded region r are the integer vectors of its closure: $S(r) = [r] \cap \mathbb{N}^n$.*

Proof. Let $v \in [r]$. Let the representation of r be given by $(a_i)_{1 \leq i \leq n}$ and $(X_i)_{0 \leq i \leq k}$. Then, for all valuations $w \in [r]$ we have:

$$\begin{aligned} \forall 0 \leq i \leq k \cdot \forall x, y \in X_i &: \langle w(x) \rangle = \langle w(y) \rangle, \\ \forall 0 \leq i < j \leq k \cdot \forall x \in X_i,\ y \in X_j &: \langle w(x) \rangle \leq \langle w(y) \rangle, \\ \forall x \in X_0 &: \langle w(x) \rangle = 0, \end{aligned}$$

Let v_0 be the valuation such that $v_0(x_i) = a_i$ for each $0 \leq i \leq k$. We have $v_0 \in [r]$ and for all $0 < j \leq k$, the valuation v_j defined by

$$\begin{aligned} v_j(x) &= v_0(x) && \text{if } x \in X_i \text{ with } i < j \\ v_j(x) &= v_0(x) + 1 && \text{if } x \in X_i \text{ with } i \geq j \end{aligned}$$

belongs to $[r]$.

We now prove that those valuations generate the whole closed region $[r]$: let w be a valuation in $[r]$. We define

$$w' = (1 - \langle w(x_k)\rangle)v_0 + (\langle w(x_1)\rangle - \langle w(x_0)\rangle)v_1 + \cdots + (\langle w(x_k)\rangle - \langle w(x_{k-1})\rangle)v_k$$

where each x_i is a clock in the corresponding X_i for $1 \leq i \leq k$ and $w(x_0) = 0$ by convention. We claim that $w' = w$. To show this, let y be a clock, and j such that $y \in X_j$. Then

$$\begin{aligned} w'(y) &= (1 - \langle w(x_k)\rangle).v_0(y) + (\langle w(x_1)\rangle - \langle w(x_0)\rangle).v_1(y) + \cdots + \\ &\quad (\langle w(x_k)\rangle - \langle w(x_{k-1})\rangle).v_k(y) \\ &= (1 - \langle w(x_k)\rangle).v_0(y) + (\langle w(x_1)\rangle - \langle w(x_0)\rangle).(v_0(y) + 1) + \cdots + \\ &\quad (\langle w(x_j)\rangle - \langle w(x_{j-1})\rangle).(v_0(y) + 1) + \\ &\quad (\langle w(x_{j+1})\rangle - \langle w(x_j)\rangle).(v_0(y)) + \cdots + \\ &\quad (\langle w(x_k)\rangle - \langle w(x_{k-1})\rangle).(v_0(y)) \\ &= v_0(y) + \langle w(x_j)\rangle = w(y) \end{aligned}$$

since $y \in X_j$. Therefore $[r] = \mathsf{Conv}(\{v_0, \ldots, v_k\})$. On the other hand, $\{v_0, \ldots, v_k\}$ is the smallest set generating $[r]$, since there is no valuation v_i that is a convex combination of the others. ■

Lemma 6.8 ensures that if a region r is a sub-region of a bounded region r', then its set of vertices $S(r)$ is the intersection of $[r]$ and the set $S(r')$ of vertices of r'.

Lemma 6.9 *Let A be a closed timed automaton. The set $\mathsf{Reach}(\llbracket A \rrbracket)$ is topologically closed.*

It is easy to prove Lemma 6.9 by showing that the successors of a closed region by a discrete transition or by the passage of time is a union of closed regions.

In the rest of this chapter, we often abusively use operators that apply to valuations v or regions r to pairs (ℓ, v) and (ℓ, r), where ℓ is a location. For example, we write $(\ell, v) + (\ell, v')$ to denote $(\ell, v + v')$ and $\lambda(\ell, v)$ to denote $(\ell, \lambda v)$, and similarly for distances, norms and neighbourhoods. We write $(\ell, v) \in (\ell, r)$ instead of $v \in r$, $[(\ell, v)]$ instead of $(\ell, [v])$, etc.

6.5 Robustness and Implementability

In his paper, Puri shows that the classical reachability analysis defined in [AD94] is not correct when the clocks may drift, even by a very small amount [Pur98]. He then reformulates the reachability problem as follows: given a timed automaton A, instead of computing $\mathsf{Reach}(\llbracket A \rrbracket_0^0)$, he proposes an algorithm that computes the limit set

$$\bigcap_{\varepsilon \in \mathbb{R}^{>0}} \mathsf{Reach}(\llbracket A \rrbracket_0^{\varepsilon})$$

When A is clear from the context, this set is denoted by R_{ε}^{*}. This is the set of states that can be reached when the clocks drift by an infinitesimally small amount. He shows that this set has nice robustness properties with respect to modeling errors. In particular, he claims that his new reachability analysis is correct when guards are subject to small imprecisions (see [Pur98] for details).

In this chapter, in order to make the link with the implementability problem, we study a variant of this robust semantics where small imprecisions on guards are allowed: the set of reachable states in this semantics is the limit set

$$\bigcap_{\Delta \in \mathbb{R}^{>0}} \mathsf{Reach}(\llbracket A \rrbracket_{\Delta}^{0})$$

When A is clear from the context, this set is abbreviated by R_{Δ}^{*}. We first show that for any timed automaton A, any zone-set Bad, we have that: $\cap_{\Delta \in \mathbb{R}^{>0}} \mathsf{Reach}(\llbracket A \rrbracket_{\Delta}^{0}) \cap \mathsf{Bad} = \varnothing$ iff there exists $\Delta \in \mathbb{R}^{>0}$ such that $\mathsf{Reach}(\llbracket A \rrbracket_{\Delta}^{0}) \cap \mathsf{Bad} = \varnothing$. Afterward, we establish that the algorithm proposed by Puri to compute the set $\cap_{\varepsilon \in \mathbb{R}^{>0}} \mathsf{Reach}(\llbracket A \rrbracket_0^{\varepsilon})$ is also valid to compute the set of states $\cap_{\Delta \in \mathbb{R}^{>0}} \mathsf{Reach}(\llbracket A \rrbracket_{\Delta}^{0})$. We also reprove the results of Puri in our broader context. This yields an algorithm for deciding the implementability problem, and proves that under assumption 6.4, both types of imprecisions have the same effect:

$$\bigcap_{\varepsilon > 0} \mathsf{Reach}(\llbracket A \rrbracket_0^{\varepsilon}) = \bigcap_{\Delta > 0} \mathsf{Reach}(\llbracket A \rrbracket_{\Delta}^{0}) = \bigcap_{\substack{\Delta > 0 \\ \varepsilon > 0}} \mathsf{Reach}(\llbracket A \rrbracket_{\Delta}^{\varepsilon})$$

For the rest of the chapter, we denote by $R_{\Delta,\varepsilon}^{*}$ the third set in the above equality.

Example Consider the closed timed automaton A_{α} of Figure 6.1 where $\alpha \in \{2, 3\}$. The automaton has two clocks x and y. There is one initial location ℓ_1 with initial predicate $x = 1 \wedge y = 0$. For location ℓ_1 and ℓ_2, the sets of reachable states in the classical semantics $\llbracket A \rrbracket$ with $\varepsilon = \Delta = 0$ are depicted in Figure 6.2. The final states (or bad states) correspond to the location err with any clock valuation. For both $\alpha = 2$ and $\alpha = 3$, the timed automaton A does not reach the bad states.

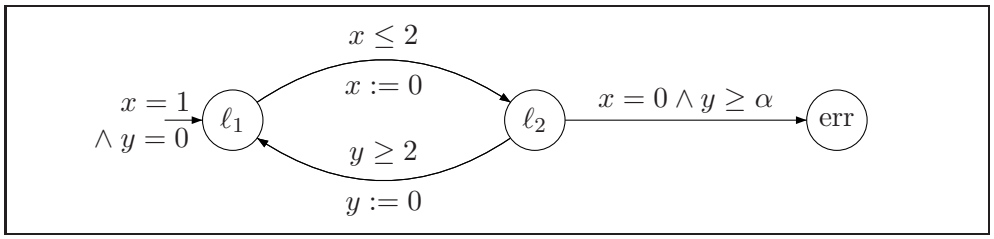


Figure 6.1: A closed timed automaton A_α with $\alpha \in \{2, 3\}$.

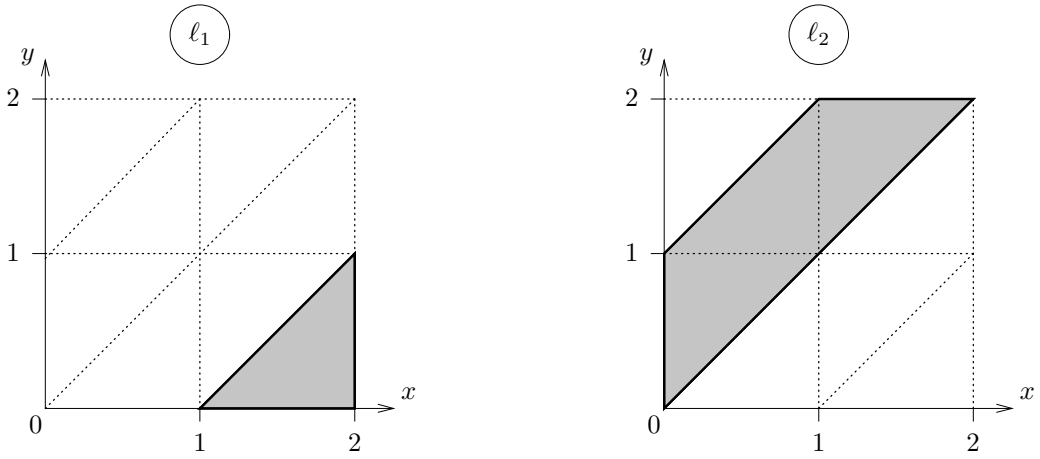


Figure 6.2: The set $\mathsf{Reach}([\![A_\alpha]\!])$ for locations ℓ_1 and ℓ_2 of A_α.

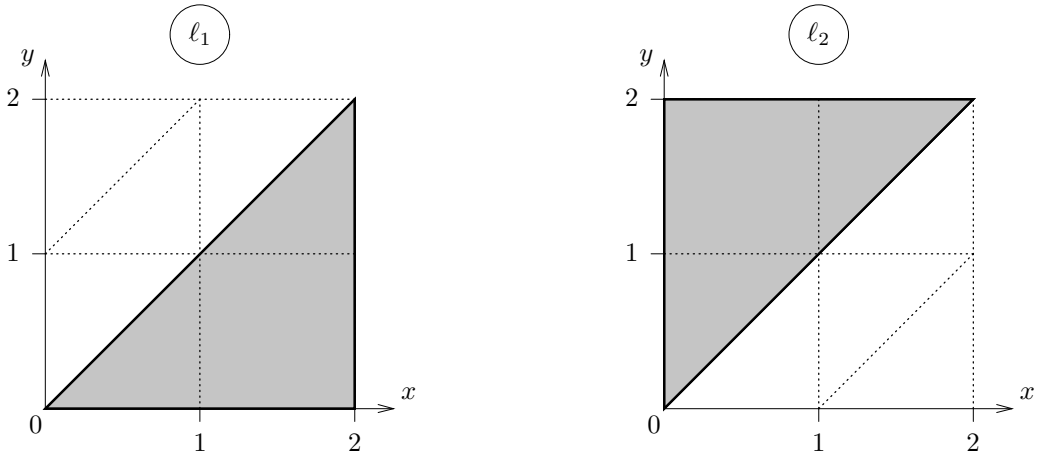


Figure 6.3: The set $R^*_\Delta = \bigcap_{\Delta>0} \mathsf{Reach}([\![A_\alpha]\!]^0_\Delta)$ for locations ℓ_1 and ℓ_2 of A_α.

Consider the enlarged semantics $[\![A]\!]^0_\Delta$ for $\varepsilon = 0$ and $\Delta > 0$. In this semantics, guards are enlarged by Δ. The edge from ℓ_1 to ℓ_2 has the guard $x \leq 2 + \Delta$ and the edge from ℓ_2 to ℓ_1 has the guard $y \geq 2 - \Delta$. From the initial state $(\ell_1, x = 1, y = 0)$, the transition to ℓ_2 can be taken Δ time units later, and so the states $(\ell_2, x = 0, y \leq 1 + \Delta)$ are reachable. Similarly, the transition from ℓ_2 back to ℓ_1 is enabled Δ time units earlier than before and the states $(\ell_1, x \geq 1 - 2\Delta, y = 0)$ are reachable. It is easy to show that after having taken k times the transitions of the cycle, the states $(\ell_1, x \geq 1 - 2k\Delta, y = 0)$ (provided $x \geq 0$) and $(\ell_2, x = 0, y \leq 1 + (2k - 1)\Delta)$ (provided $y \leq 2$) are reachable. Hence, for all $\Delta > 0$ the states $(\ell_1, x \geq 0, y = 0)$ and $(\ell_2, x = 0, y \leq 2)$ are reachable in $[\![A]\!]^0_\Delta$ and were not reachable in the classical semantics. Those states are represented in Figure 6.3.

The same situation occurs in the enlarged semantics $[\![A]\!]^\varepsilon_0$ for $\Delta = 0$ and $\varepsilon > 0$. The set of states that are reachable no matter the value of $\varepsilon > 0$ is the same as on Figure 6.3.

This example shows that that $\mathsf{Reach}([\![A]\!]) \neq R^*_\Delta$ and $\mathsf{Reach}([\![A]\!]) \neq R^*_\varepsilon$. On the other hand, it shows that the classical semantics is not robust with regard to small perturbations in either the timing constraints or the clock rate. The effect of such perturbations, no matter how small they are, may lead to dramatically different behaviours of the system. In this case, the location err is reachable in the perturbed version of $A_{\alpha=2}$ but not in the classical semantics. We say that the safety property (to avoid the location err) is not robustly satisfied by $A_{\alpha=2}$. On the other hand, for $\alpha = 3$ the safety property still holds in the limit of the enlarged semantics. As we will demonstrate, this implies that there exists a strictly positive value for Δ such that the enlarged semantics $[\![A]\!]^0_\Delta$ is safe. In fact, any $\Delta < \frac{1}{3}$ fits.

The main result of this chapter is Theorem 6.10.

Theorem 6.10 *Under assumption 6.4, there exists an algorithm that decides, given an Alur-Dill automaton A and a bounded zone-set* Bad*, whether there exist $\Delta \in \mathbb{R}^{>0}$ such that* $\mathsf{Reach}([\![A]\!]^0_\Delta) \cap \mathsf{Bad} = \varnothing$.

To prove Theorem 6.10, we show that the algorithm proposed by Puri to compute R^*_ε is correct and also computes R^*_Δ. As a corollary, the two sets are equal. That proof is given in the next section. The connection with the implementability problem is established by the following theorem.

Theorem 6.11 *Under assumption 6.4, for all Alur-Dill automata A, and all bounded zone-sets* Bad*, the following equivalence holds:*

$$R^*_\Delta \cap \mathsf{Bad} = \varnothing \qquad \textit{iff} \qquad \exists \Delta > 0 : \mathsf{Reach}([\![A]\!]^0_\Delta) \cap \mathsf{Bad} = \varnothing.$$

The proof of Theorem 6.11 is based on the following lemmas. The first one corrects a wrong claim of Puri about a lower bound on the ∞-distance between two zones with

empty intersection. This bound is claimed to be $\frac{1}{2}$ in [Pur98, Lemma 6.4]. We show that $\frac{1}{n}$ is the tightest bound, where n is the dimension of the space .

Lemma 6.12 *Let $Z_1, Z_2 \subseteq \mathbb{R}^n$ be two zones such that $Z_1 \cap Z_2 = \varnothing$. For all $x \in Z_1$ and $y \in Z_2$, we have $d_\infty(x,y) \geq \frac{1}{n}$. This bound is tight.*

Proof. Let us recall the definition of the 1-norm and the ∞-norm:

$$\|x\|_\infty = \max_{1\leq i\leq n}(|x_i|) \qquad\qquad \|x\|_1 = \sum_{i=1}^{n}|x_i|$$

First, we show that $\frac{1}{n}$ is a lower bound. Clearly, for all $v \in \mathbb{R}^n$, $\|v\|_1 \leq n \cdot \|v\|_\infty$. We prove that $\|x-y\|_1 \geq 1$, which entails the result.

We consider two zones given by two DBM in normal form: $Z_1 \equiv [\![(m_{i,j})]\!]$ and $Z_2 \equiv [\![(m'_{i,j})]\!]$. Since $Z_1 \cap Z_2 = \varnothing$, there must exist a "negative cycle":

$$m^{(\prime)}_{i_1,i_2} + m^{(\prime)}_{i_2,i_3} + m^{(\prime)}_{i_3,i_4} + \cdots + m^{(\prime)}_{i_p,i_1} \leq -1$$

where each term $m^{(\prime)}_{i,j}$ of the sum can be taken either in the matrix of Z_1 or in the matrix of Z_2. We may assume that at least one $m^{(\prime)}_{i,j}$ comes from Z_1 and one from Z_2 since otherwise Z_1 (or Z_2) would be empty and the result holds vacuously.

Since for DBM in normal form, we have $m_{a,b} + m_{b,c} \geq m_{a,c}$ for all indices a, b, c, we can merge any two *consecutive* $m_{i,j}$'s into one while keeping the inequality. The same holds for $m'_{i',j'}$, and we can thus assume that $m_{i,j}$ and $m'_{i',j'}$ alternate in the sum above (starting with m_{i_1,i_2}, say).

Pick $x \in Z_1$ and $y \in Z_2$. Then

$$(x_{i_2} - x_{i_1}) + (y_{i_3} - y_{i_2}) + (x_{i_4} - x_{i_3}) + \cdots + (y_{i_1} - y_{i_p}) \leq -1.$$

Terms can be rearranged in this sum, yielding

$$(y_{i_1} - x_{i_1}) - (y_{i_2} - x_{i_2}) + (y_{i_3} - x_{i_3}) - \cdots - (y_{i_p} - x_{i_p}) \leq -1.$$

If $i_k = 0$ for some k, then $x_{i_k} - y_{i_k} = 0$. Thus, we assume that $1 \leq i_k \leq n$. We take the absolute value, and apply the triangle inequality:

$$\begin{aligned} 1 &\leq \big|(y_{i_1} - x_{i_1}) - (y_{i_2} - x_{i_2}) + (y_{i_3} - x_{i_3}) - \cdots - (y_{i_p} - x_{i_p})\big| \\ &\leq |(y_{i_1} - x_{i_1})| + |(y_{i_2} - x_{i_2})| + |(y_{i_3} - x_{i_3})| + \cdots + |(y_{i_p} - x_{i_p})| \\ &\leq \|x-y\|_1. \end{aligned}$$

Now, let us show that this bound is tight. Consider the zones $Z_1, Z_2 \subseteq \mathbb{R}^n$ defined by the following equations:

- If n is odd

$$Z_1 \equiv \begin{cases} x_1 = 1 \\ x_{2i} - x_{2i+1} = 0 \qquad 1 \leq i \leq \frac{n-1}{2} \end{cases}$$

$$Z_2 \equiv \begin{cases} x_{2i-1} - x_{2i} = 0 \qquad 1 \leq i \leq \frac{n-1}{2} \\ x_n = 0 \end{cases}$$

- If n is even

$$Z_1 \equiv \begin{cases} x_1 = 1 \\ x_{2i} - x_{2i+1} = 0 \qquad 1 \leq i \leq \frac{n}{2} - 1 \\ x_n = 0 \end{cases}$$

$$Z_2 \equiv \begin{cases} x_{2i-1} - x_{2i} = 0 \qquad 1 \leq i \leq \frac{n}{2} \end{cases}$$

We have $Z_1 \cap Z_2 = \varnothing$; indeed combining the equations of Z_1 and Z_2 yields $x_i = x_j$ for all $0 \leq i, j \leq n$, which leads to a contradiction since $x_1 = 1$ and $x_n = 0$. On the other hand, let $p = (1, \frac{n-2}{n}, \frac{n-2}{n}, \frac{n-4}{n}, \frac{n-4}{n}, \dots)$ and $q = (\frac{n-1}{n}, \frac{n-1}{n}, \frac{n-3}{n}, \frac{n-3}{n}, \frac{n-5}{n}, \dots)$ (take the first n coordinates). It is easy to check that $p \in Z_1$ and $q \in Z_2$, while $d_\infty(p, q) = \max(\frac{1}{n}, \dots, \frac{1}{n}) = \frac{1}{n}$. ■

The next three lemmas are purely related to set theory and topology. For Lemma 6.13 and Lemma 6.14, the proof is immediate, but for Lemma 6.15, the argument is more involved.

Lemma 6.13 *If $d_\infty(A, B) > 0$, then $A \cap B = \varnothing$.*

Lemma 6.14 *If $A \subseteq B$, then $d_\infty(A, C) \geq d_\infty(B, C)$ for all C.*

Lemma 6.15 *Let $A_\Delta (\Delta \in \mathbb{R}^{>0})$ be a collection of closed sets such that $A_{\Delta_1} \subseteq A_{\Delta_2}$ if $\Delta_1 \leq \Delta_2$. Let $A = \bigcap_{\Delta > 0} A_\Delta$ be nonempty. If B is a bounded set with $d_\infty(A, B) > 0$, then there exists $\Delta > 0$ such that $A_\Delta \cap B = \varnothing$.*

Proof. We proceed by contradiction. Assume that for all $\Delta > 0$, we have $A_\Delta \cap B \neq \varnothing$. Let $\delta_i = \frac{1}{i}$ (for each $i \geq 1$). Then, we have:

$$\forall i \geq 1 \cdot \exists x_i \in A_{\delta_i} \cap B$$

Since B is bounded, so is the set $\{x_i \mid i \geq 1\}$. By Bolzano-Weierstrass Theorem, there exists a point x such that:

$$\forall \epsilon > 0 \cdot \forall i \geq 1 \cdot \exists j \geq i : x_j \in \mathcal{N}_2(x, \epsilon) \cap A_{\delta_j}$$

Let us show that x is in the closure of A_{δ_i} for all $i \geq 1$. Since $A_{\delta_j} \subseteq A_{\delta_i}$ for all $j \geq i$, we have:

$$\forall i \geq 1 \cdot \forall \epsilon > 0 \cdot \exists j : x_j \in \mathcal{N}_2(x, \epsilon) \cap A_{\delta_i}$$

that is:

$$\forall i \geq 1 \cdot \forall \epsilon > 0 : \mathcal{N}_2(x, \epsilon) \cap A_{\delta_i} \neq \varnothing$$

Hence, for all $i \geq 1$ the point x is in the closure of the closed set A_{δ_i}, and thus $x \in A_{\delta_i}$. Since, for all $\Delta > 0$ there exists $i \geq 1$ such that $\delta_i \leq \Delta$ and thus $A_{\delta_i} \subseteq A_\Delta$, we have $\forall \Delta \in \mathbb{R}^{>0} : x \in A_\Delta$. This entails that $x \in A$. Now observe that:

$$\forall i \geq 1 : d_\infty(\{x\}, B) \leq d_\infty(x, x_i) + d_\infty(\{x_i\}, B)$$

Observe that $d_\infty(x, x_i)$ can be made arbitrarily small for sufficiently large i, and $d_\infty(\{x_i\}, B) = 0$ since $x_i \in B$ for all $i \geq 1$. Therefore, we get:

$$\forall \epsilon > 0 : d_\infty(\{x\}, B) \leq \epsilon$$

and thus $d_\infty(\{x\}, B) = 0$ which is a contradiction. ■

Now, we present the proof of Theorem 6.11.

Proof of Theorem 6.11. First, assume that $R^*_\Delta \cap \mathsf{Bad} = \varnothing$. Since R^*_Δ and Bad are zone-sets[2], by Lemma 6.12 we have $d_\infty(R^*_\Delta, \mathsf{Bad}) > 0$. From Lemma 6.15, we obtain that there exists $\Delta > 0$ such that $\mathsf{Reach}([\![A]\!]^0_\Delta) \cap \mathsf{Bad} = \varnothing$.

Second, assume that there exists $\Delta > 0$ such that $\mathsf{Reach}([\![A]\!]^0_\Delta) \cap \mathsf{Bad} = \varnothing$. Then trivially $R^*_\Delta \cap \mathsf{Bad} = \varnothing$. ■

Results similar to Theorem 6.11 can be proven in the same way, namely:

$$\begin{aligned} R^*_\varepsilon \cap \mathsf{Bad} = \varnothing \;\; &\text{iff} \;\; \exists \varepsilon > 0 : \mathsf{Reach}([\![A]\!]^\varepsilon_0) \cap \mathsf{Bad} = \varnothing \\ \text{and } R^*_{\Delta,\varepsilon} \cap \mathsf{Bad} = \varnothing \;\; &\text{iff} \;\; \exists \Delta, \varepsilon > 0 : \mathsf{Reach}([\![A]\!]^\varepsilon_\Delta) \cap \mathsf{Bad} = \varnothing. \end{aligned}$$

6.6 Algorithm for computing R^*_Δ, R^*_ε and $R^*_{\Delta,\varepsilon}$

In this section, we prove that $R^*_\varepsilon = R^*_\Delta = R^*_{\Delta,\varepsilon}$, and that those sets are computed by Algorithm 4 (originally proposed in [Pur98]). To help the reader to follow the further developments, we give in Figure 6.6 a reminder of the main steps of the proof, and we explain informally the principle of the algorithm on a simple example.

Example Consider the closed timed automaton A_α of Figure 6.1 (the value of α does not matter here). The reachable states of A in locations ℓ_1 and ℓ_2 are depicted on Figure 6.2 and are computed in J^* by the algorithm at line 4. Then, in the while-loop, the algorithm adds to J^* the progress cycles of the region automaton of A that "touch"

[2]The set J^* computed by Algorithm 4 below is a zone-set, and we show in the correctness proof of the algorithm that $J^* = R^*_\Delta$.

Algorithm 4: Algorithm for computing the limit sets R^*_Δ, R^*_ε and $R^*_{\Delta,\varepsilon}$ of a closed timed automaton A.

Data: A closed timed automaton $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$.

Result: The set $J^* = R^*_\varepsilon = R^*_\Delta = R^*_{\Delta,\varepsilon}$.

begin

1 Construct the bounded region automaton R_A of A ;

2 Compute the set $\mathsf{PC}(R_A)$ of progress cycles of R_A ;

3 $Q_0 \leftarrow \{(\ell, r) \mid \ell \in \mathsf{Loc} \wedge r \in \mathcal{R}_A \wedge r \models \mathsf{Init}(\ell)\}$;

4 $J^* \leftarrow \mathsf{Reach}(R_A, Q_0)$;

5 **while** for some $S = p_0\, p_1\, \ldots\, p_k \in \mathsf{PC}(R_A)$, $[p_0] \not\subseteq J^*$ and $J^* \cap [p_0] \neq \varnothing$ **do**

6 $\quad J^* \leftarrow J^* \cup [p_0]$;

7 $\quad J^* \leftarrow \mathsf{Reach}(R_A, J^*)$;

8 **return** J^* ;

end

the set J^*, and performs a reachability analysis from the new states in the classical semantics. In the example, the progress cycle (ℓ_1, R_1), (ℓ_1, R_2), (ℓ_1, R_3), (ℓ_2, R_4) shown in Figure 6.4 is added, and the set J^* computed by the algorithm is the set R^*_Δ shown in Figure 6.3.

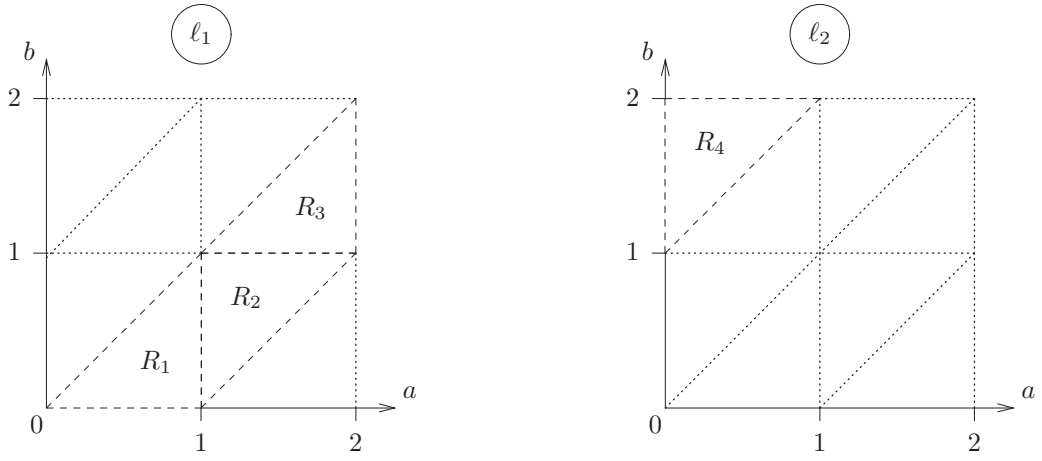


Figure 6.4: A progress cycle R_1, R_2, R_3, R_4 in the region automaton of A_α.

To prove the correctness of Algorithm 4, we first study the properties of limit cycles.

6.6.1 Limit cycles

Definition 6.16 [Limit Cycle] A *limit cycle* of a closed timed automaton A is a finite trajectory π of $[\![A]\!]$ that contains at least one discrete transition and such that $\mathsf{last}(\pi) = \mathsf{first}(\pi)$. □

As suggested in [Pur98], given a progress cycle in the bounded region automaton and a region on this cycle, only a subset of the points of the region have a limit cycle.

Definition 6.17 Given a path $p = p_0\, p_1 \cdots p_N$ in the bounded region automaton of a timed automaton A, we say that a trajectory π of $[\![A]\!]$ *follows* p if $|\pi| = N$ and for all i, $0 \leq i \leq N$, $\mathsf{state}_i(\pi) \in [p_i]$. □

Definition 6.18 Let $p = p_0\, p_1 \ldots p_N$ be a cycle in the bounded region automaton of a closed timed automaton A (thus such that $p_N = p_0$). For $Q_0 \subseteq [p_0]$, define the *return map* $R_p(Q_0)$ as follows:

$$R_p(Q_0) = \left\{ q \in [p_N] \;\middle|\; \begin{array}{l} \text{there exists a trajectory } \pi \text{ of } [\![A]\!] \text{ that follows } p \\ \text{such that } \mathsf{first}(\pi) \in Q_0 \text{ and } \mathsf{last}(\pi) = q \end{array} \right\}$$

For $i \geq 2$, define recursively $R_p^i(Q_0) = R_p(R_p^{i-1}(Q_0))$ and let $L_{i,p}$ be the set of points that can return back to themselves after i cycles through p: $L_{i,p} = \{q \mid q \in R_p^i(\{q\})\}$. We write $L_p = \cup_{i \in \mathbb{N}^{>0}} L_{i,p}$. □

The following key property of L_p is central to the proof of correctness of Algorithm 4. It states that L_p is both forward and backward reachable from all valuation in a cycle p.

Theorem 6.19 ([Pur98, Lemma 7.10**])** *Let $p = p_0 \ldots p_N$ be a cycle in the bounded region automaton of a closed timed automaton. For all $z \in [p_0]$, there exists $z', z'' \in L_p$ and trajectories π, π' in $[\![A]\!]$ such that*

- $\mathsf{first}(\pi) = z$ *and* $\mathsf{last}(\pi) = z'$ *and*
- $\mathsf{first}(\pi') = z''$ *and* $\mathsf{last}(\pi') = z$.

The proof proposed by Puri is quite sketchy. We propose a complete proof that uses the following intermediate lemmas.

Lemma 6.20 ([Pur98, Lemma 7.1**])** *Let $p = p_0\, p_1 \ldots p_N$ be a path in the bounded region automaton of a closed timed automaton A, let π and π' be two trajectories of $[\![A]\!]$ that follow p. Then for all $\lambda \in [0, 1]$, there exists a trajectory π'' of $[\![A]\!]$ that follows p and such that $\mathsf{first}(\pi'') = \lambda.\mathsf{first}(\pi) + (1 - \lambda).\mathsf{first}(\pi')$ and $\mathsf{last}(\pi'') = \lambda.\mathsf{last}(\pi) + (1 - \lambda).\mathsf{last}(\pi')$.*

Proof. Let $\pi = (q_0, t_0)\,\sigma_1\,(q_1, t_1)\,\sigma_2 \cdots \sigma_N\,(q_N, t_N)$ and $\pi' = (q'_0, t'_0)\,\sigma'_1 \cdots \sigma'_N\,(q'_N, t'_N)$. Consider the sequence

$$\pi'' = (q''_0, t''_0)\,\sigma''_1\,(q''_1, t''_1)\,\sigma''_2 \cdots \sigma''_N\,(q''_N, t''_N)$$

where for all $0 \leq i \leq N$, $q''_i = \lambda.q_i + (1-\lambda).q'_i$ and $t''_i = \lambda.t_i + (1-\lambda).t'_i$ and for all $1 \leq i \leq N$, $\sigma''_i = \lambda.\sigma_i + (1-\lambda).\sigma'_i$ if $\sigma_i \in \mathbb{T}$ and $\sigma''_i = \sigma_i$ otherwise. It is easy to show that π'' is a trajectory in $[\![A]\!]$ since regions are convex sets. ■

Lemma 6.21 ([Pur98, Lemma 7.3**])** *Let p be a cycle in the bounded region automaton of a closed timed automaton. Then L_p is convex.*

Proof. Let $x, y \in L_p$, and $\lambda \in [0,1]$. There exists natural numbers k and l such that $x \in L_{k,p}$ and $y \in L_{l,p}$. Then $x, y \in L_{k.l,p}$, and according to Lemma 6.20, we have $\lambda.x + (1-\lambda).y \in L_{k.l,p} \subseteq L_p$. ■

Definition 6.22 Let $p = p_0\,p_1\,\ldots\,p_N$ be a cycle in the bounded region automaton of a closed timed automaton (thus $p_0 = p_N$). The *orbit graph* is the graph $\Theta_p = (V_\Theta, \to_\Theta)$ such that $V_\Theta = S(p_0)$ is the set of vertices of p_0 and for all $v, w \in V_\Theta$, $v \to_\Theta w$ iff $w \in R_p(\{v\})$. For $m \in \mathbb{N}$ and $v \in V_\Theta$, we define

$$\mathsf{Succ}^m(v) = \{w \in V_\Theta \mid v \to^m_\Theta w\} \quad \text{and} \quad \mathsf{Pred}^m(v) = \{w \in V_\Theta \mid w \to^m_\Theta v\}.$$

□

Given a vertex $v \in V_\Theta$, the set $R_p(\{v\})$ is a closed region according to Lemma 6.9, and thus we have $R_p(\{v\}) = \mathsf{Conv}(\{w \in V_\Theta \mid v \to_\Theta w\})$ as a closed region contains all its vertices. More generally, we have $R^k_p(\{v\}) = \mathsf{Conv}(\{w \in V_\Theta \mid v \to^k_\Theta w\})$ for all $k \geq 1$.

Lemma 6.23 ([Pur98, Lemma 7.4**])** *If there exists a path $p \ldots p'$ in the bounded region automaton of a closed timed automaton A, then for all vertices $v \in S(p)$ of p, there exists a vertex $v' \in S(p')$ of p' such that there exists a trajectory of $[\![A]\!]$ from v to v', and conversely, for all vertices $v' \in S(p')$ of p', there exists a vertex $v \in S(p)$ of p such that there exists a trajectory of $[\![A]\!]$ from v to v'.*

Proof. Let v be a vertex of $[p]$. Then $\{v\}$ forms a subregion of $[p]$, and its successors in $[p']$ form a closed subregion of $[p']$. According to Lemma 6.8, that subregion of $[p']$ necessarily contains a vertex.

The same argument can be applied backward, since the predecessor of a subregion of p' is a closed subregion of p. ■

Proof of Theorem 6.19. Let $\Theta_p = (V_\Theta, \to_\Theta)$ be the orbit graph of p.

Let $V = \{v \in V_\Theta \mid \exists m \in \mathbb{N}.\ v \in \mathsf{Succ}^m(v)\}$. Lemma 6.23 entails that every vertex in the orbit graph has an outgoing edge. Thus for all $v \in V_\Theta$, there exists an integer m_v such that $\forall m \geq m_v : \mathsf{Succ}^m(v) \cap V \neq \varnothing$, because V_Θ is finite. Let $M = \max\{m_v \mid v \in V_\Theta\}$ be the largest such m_v. Then $\mathsf{Succ}^M(v) \cap V \neq \varnothing$ for all v. A similar argument proves the existence of M' such that $\mathsf{Pred}^{M'}(v) \cap V \neq \varnothing$ for all v.

Since $z \in [p_0]$, we can write $z = \sum_i \lambda_i v_i$, where $\lambda_i \in [0,1]$, $\sum_i \lambda_i = 1$ and $v_i \in V_\Theta$. For each v_i, let w_i be an element of $\mathsf{Succ}^M(v_i) \cap V$. From Lemma 6.20, there is a path from z to $z' = \sum_i \lambda_i w_i$ and $z' \in \mathsf{Conv}(V)$. By Lemma 6.21 we have $\mathsf{Conv}(V) \subseteq L$ and thus $z' \in L$.

Conversely, if x_i is a vertex in $\mathsf{Pred}^{M'}(v_i) \cap V$, there is a path from $z'' = \sum_i \lambda_i x_i \in \mathsf{Conv}(V) \subseteq L$ to z. ■

6.6.2 Soundness of Algorithm 4: $J^* \subseteq R^*_\Delta$ and $J^* \subseteq R^*_\varepsilon$

We show that the set J^* computed by Algorithm 4 is reachable in the limit sets R^*_Δ and R^*_ε. In particular, for all progress cycles that are added to J^* by the algorithm, we show that every point of the cycle is reachable if either a drift on clocks or an enlargement of the guards is allowed, no matter how small it is.

The proof is based on Theorem 6.19 and on the fact that for all progress cycles p, the set L_p is a strongly connected component of both $[\![A]\!]^0_\Delta$ and $[\![A]\!]^\varepsilon_0$ for all $\Delta, \varepsilon > 0$. Hence in L_p, every state is reachable from every state for the enlarged semantics, and thus similarly, in each region of the cycle p every state is reachable from every state by Theorem 6.19.

6.6.2.1 Imprecise guards: $J^* \subseteq R^*_\Delta$.

Theorem 6.24 *Let A be a closed timed automaton, let $p = p_0\, p_1 \cdots p_N$ be a progress cycle of the bounded region automaton of A, and $\Delta \in \mathbb{R}^{>0}$. For all states $u, v \in L_p$, there exists a trajectory π of $[\![A]\!]^0_\Delta$ such that $\mathsf{first}(\pi) = u$ and $\mathsf{last}(\pi) = v$.*

This theorem results immediately from the following Lemma.

Lemma 6.25 *Let A be a closed timed automaton, let $p = p_0\, p_1 \cdots p_N$ be a progress cycle of the bounded region automaton of A. For all $\Delta \in \mathbb{R}^{>0}$, for all state $u \in L_p$ and for all neighbour state $v \in [p_0] \cap \mathcal{N}_\infty(u, \frac{\Delta}{2})$, there exists a trajectory π' of $[\![A]\!]^0_\Delta$ such that $\mathsf{first}(\pi') = u$ and $\mathsf{last}(\pi') = v$.*

Proof. Let $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$. Since $u \in L_p$, there exists a trajectory π of $[\![A]\!]^0_0$ that follows p a certain number of times and such that $\mathsf{first}(\pi) =$

$\mathsf{last}(\pi) = u$. We slightly modify π such that timed and discrete transitions alternate (we insert a zero length timed transition between two consecutive discrete transitions, and we merge consecutive timed transitions). Assume that[3]:

$$\pi = (\ell_0, u_0) \xrightarrow{t_0} (\ell_0, u'_0) \xrightarrow[R_0]{\sigma_0} (\ell_1, u_1) \xrightarrow{t_1} (\ell_1, u'_1) \xrightarrow[R_1]{\sigma_1} \cdots$$
$$\cdots \xrightarrow[R_{m-2}]{\sigma_{m-2}} (\ell_{m-1}, u_{m-1}) \xrightarrow{t_{m-1}} (\ell_{m-1}, u'_{m-1}) \xrightarrow[R_{m-1}]{\sigma_{m-1}} (\ell_m, u_m) \xrightarrow{t_m} (\ell_m, u'_m)$$

with $u_0 = u'_m = u$. Each $t_i \in \mathbb{R}^{\geq 0}$ and $\sigma_i \in \Sigma$. We annotate π with sets of clocks $R_i \subseteq \mathsf{Var}$ that are reset by discrete transitions σ_i. Note that $\bigcup_{i=0}^{i=m-1} R_i = \mathsf{Var}$.

Intuitively, we prove the lemma by modifying the length of the timed transitions of π so that the clocks are reset slightly earlier or later than in π. We obtain a trajectory of $[\![A]\!]^0_\Delta$ because the guards are enlarged and therefore they are enabled in the states of the new trajectory.

Let the representation of p_0 be given by $(a_x)_{x\in\mathsf{Var}}$ and $(X_i)_{0\leq i\leq k}$. For all valuations $w \in p_0$, we have:

- for all $x \in \mathsf{Var}$: $\lfloor w(x) \rfloor = a_x$;
- for all $x \in X_0$: $\langle w(x) \rangle = 0$;
- for all i and for all $x, y \in X_i$: $\langle w(x) \rangle = \langle w(y) \rangle$;
- for all $i < j$ and for all $x \in X_i,\ y \in X_j$: $\langle w(x) \rangle < \langle w(y) \rangle$;

Since p is a progress cycle, we know that each clock is reset at least once along π. For each clock $x \in \mathsf{Var}$, let α_x be the index of the last transition of π in which x is reset. Formally, we have:

$$x \in R_{\alpha_x} \qquad\qquad \forall i > \alpha_x : x \notin R_i \tag{6.1}$$

Then, for each clock $x \in \mathsf{Var}$, we have:

$$u_0(x) = u'_m(x) = u(x) = \sum_{i=\alpha_x+1}^{m} t_i \tag{6.2}$$

Let $v \in [p_0] \cap \mathcal{N}_\infty(u, \frac{\Delta}{2})$ and for each $x \in \mathsf{Var}$ let $\delta_x = v(x) - u(x)$. Clearly, we have $|\delta_x| \leq \frac{\Delta}{2}$. Moreover since $v \in [p_0]$, the closed version of the above inequalities defining p_0 are satisfied by v. Let $\langle\!\langle v(x) \rangle\!\rangle = v(x) - a_x$, we have:

[3]It is not restrictive to assume that π starts and ends with a timed transition as zero length timed transitions are possible.

- for all $x \in \mathsf{Var}$: $0 \leq \langle\!\langle v(x) \rangle\!\rangle \leq 1$;
- for all $x \in X_0$: $\langle\!\langle v(x) \rangle\!\rangle = 0$;
- for all i and for all $x, y \in X_i$: $\langle\!\langle v(x) \rangle\!\rangle = \langle\!\langle v(y) \rangle\!\rangle$;
- for all $i < j$ and for all $x \in X_i,\ y \in X_j$: $\langle\!\langle v(x) \rangle\!\rangle \leq \langle\!\langle v(y) \rangle\!\rangle$;

This entails that:

- for all i and for all $x, y \in X_i$: $\delta_x = \langle\!\langle v(x) \rangle\!\rangle - \langle u(x) \rangle = \langle\!\langle v(y) \rangle\!\rangle - \langle u(y) \rangle = \delta_y$;
- for all $x, y \in \mathsf{Var}$ such that $u(x) < u(y)$ (and hence $\alpha_x > \alpha_y$ from Equation (6.2)), we have $v(x) \leq v(y)$ and thus $u(x) + \delta_x \leq u(y) + \delta_y$, that is:

$$\delta_x - \delta_y \leq u(y) - u(x) = \sum_{i=\alpha_y+1}^{\alpha_x} t_i \tag{6.3}$$

Let $\Gamma = \{\alpha_x \mid x \in \mathsf{Var}\} = \{\alpha_1, \ldots, \alpha_l\}$ be the set of positions in π where a clock is reset for the last time. Assume without loss of generality that $\alpha_1 < \alpha_2 < \cdots < \alpha_l$ and that for all $1 \leq i \leq l$, the clock $x_i \in \mathsf{Var}$ is such that that $\alpha_{x_i} = \alpha_i$. Consider the time stamps in π as the following block-sequence, and construct the sequence $(t'_i)_{0 \leq i \leq m}$ by adding a *shift* given as follows:

$$\begin{array}{c} \big[t_0 \ldots t_{\alpha_1}\big]\big[t_{\alpha_1+1} \ldots t_{\alpha_2}\big] \ldots \big[t_{\alpha_{j-1}+1} \ldots t_{\alpha_j}\big] \ldots \big[t_{\alpha_{l-1}+1} \ldots t_{\alpha_l}\big]\big[t_{\alpha_l+1} \ldots t_m\big] \\ +0 \qquad +\delta_1 - \delta_2 \quad \ldots \quad +\delta_{j-1} - \delta_j \quad \ldots \quad +\delta_{l-1} - \delta_l \qquad +\delta_l \\ \hline = \quad \big[t'_0 \ldots t'_{\alpha_1}\big]\big[t'_{\alpha_1+1} \ldots t'_{\alpha_2}\big] \ldots \big[t'_{\alpha_{j-1}+1} \ldots t'_{\alpha_j}\big] \ldots \big[t'_{\alpha_{l-1}+1} \ldots t'_{\alpha_l}\big]\big[t'_{\alpha_l+1} \ldots t'_m\big] \end{array}$$

where each t'_i is obtained from t_i by distributing the shift of each block over the time stamps of the block. This can be done such that each t'_i is nonnegative for all $0 \leq i \leq m$ since for all $i \leq \alpha_1$ we have $t'_i = t_i$, for all $2 \leq j \leq l$ we have:

$$\sum_{i=\alpha_{j-1}+1}^{\alpha_j} t'_i \ = \left(\sum_{i=\alpha_{j-1}+1}^{\alpha_j} t_i \right) + \delta_{j-1} - \delta_j \geq 0 \quad \text{by Equation (6.3)},$$

and finally for all $i \geq \alpha_l + 1$ we have:

$$\sum_{i=\alpha_l+1}^{m} t'_i \ = \sum_{i=\alpha_l+1}^{m} t_i + \delta_l = u(x_l) + \delta_l = v(x_l) \geq 0$$

We now construct the trajectory π' from π by simply replacing each t_i by t'_i:

$$\pi' = (\ell_0, v_0) \xrightarrow{t'_0} (\ell_0, v'_0) \xrightarrow[R_0]{\sigma_0} (\ell_1, v_1) \xrightarrow{t'_1} (\ell_1, v'_1) \xrightarrow[R_1]{\sigma_1} \cdots$$
$$\cdots \xrightarrow[R_{m-2}]{\sigma_{m-2}} (\ell_{m-1}, v_{m-1}) \xrightarrow{t'_{m-1}} (\ell_{m-1}, v'_{m-1}) \xrightarrow[R_{m-1}]{\sigma_{m-1}} (\ell_m, v_m) \xrightarrow{t'_m} (\ell_m, v'_m)$$

where $v_0 = u_0 = u$, for all $0 \le i \le m : v'_i = v_i + t'_i$ and for all $1 \le i \le m : v_i = v'_{i-1}[R_{i-1} := 0]$. We claim that π' is a trajectory of $[\![A]\!]^0_\Delta$. To show this, we must verify that the guard (which is enlarged by Δ in $[\![A]\!]^0_\Delta$) of each discrete transition σ_i is satisfied by v'_i. Since π is a trajectory of $[\![A]\!]$, we know that each u'_i satisfies the corresponding guard under the classical semantics. Therefore, it is sufficient to prove that the difference $|u'_i(x) - v'_i(x)|$ is bounded by Δ for all $x \in \mathsf{Var}$. To do that, let j be the greatest index such that $j \le i$ and $u_j(x) = v_j(x)$ (such an index exists because $u_0 = v_0$). Clearly, the difference $|u'_i(x) - v'_i(x)|$ is bounded by the sum of the shifts that we have introduced between index j and i. For uniformity, let the first shift be $\delta_0 - \delta_1$ with $\delta_0 = \delta_1$ and the last shift be $\delta_l - \delta_{l+1}$ with $\delta_{l+1} = 0$. Notice that $|\delta_i| \le \frac{\Delta}{2}$ holds for all $i = 0, \ldots, l+1$. If a block $\left[t_{\alpha_{p-1}+1} \ldots t_{\alpha_p}\right]$ is such that $j \le \alpha_{p-1} + 1$ and $\alpha_p \le i$, then the whole shift $\delta_{p-1} - \delta_p$ counts in the sum. On the other hand, if i or j lies inside the block, then only a portion $\alpha(\delta_{p-1} - \delta_p)$ of the shift counts where $\alpha \in [0, 1]$. Accordingly, the sum of the shifts can take one of the three forms (where $\alpha, \beta \in [0, 1]$):

- $s_1 = \alpha(\delta_p - \delta_{p+1})$ (if i and j lie in the same block)
- $s_2 = \alpha(\delta_p - \delta_{p+1}) + \beta(\delta_{p+1} - \delta_{p+2})$ (if i and j lie in consecutive blocks)
- $s_3 = \alpha(\delta_p - \delta_{p+1}) + \delta_{p+1} - \delta_q + \beta(\delta_q - \delta_{q+1})$ (otherwise)

It is easy to show the following bounds:

- $|s_1| \le \alpha \cdot 2 \cdot \frac{\Delta}{2} \le \Delta$
- $\left.\begin{array}{l} |s_2| = |\alpha(\delta_p - \delta_{p+2}) + (\beta - \alpha)(\delta_{p+1} - \delta_{p+2})| \le \alpha \cdot \Delta + |\beta - \alpha|\Delta \\ |s_2| = |(\alpha - \beta)(\delta_p - \delta_{p+1}) + \beta(\delta_p - \delta_{p+2})| \le |\alpha - \beta|\Delta + \beta \cdot \Delta \end{array}\right\} \Rightarrow |s_2| \le \Delta$
- $\left.\begin{array}{l} |s_3| = |\alpha(\delta_p - \delta_{q+1}) + (1 - \beta)(\delta_{p+1} - \delta_q) + (\beta - \alpha)(\delta_{p+1} - \delta_{q+1})| \\ |s_3| = |(\alpha - \beta)(\delta_p - \delta_q) + (1 - \alpha)(\delta_{p+1} - \delta_q) + \beta(\delta_p - \delta_{q+1})| \end{array}\right\} \Rightarrow |s_3| \le \Delta$

which shows that $|u'_i(x) - v'_i(x)| \le \Delta$ for all $x \in \mathsf{Var}$.

Finally, since the sets of clocks R_i that are reset in π' are the same as in π, Equation (6.2) applies and we get for all $x \in \mathsf{Var}$:

$$v'_m(x) \;=\; \sum_{i=\alpha_x+1}^{m} t'_i \;=\; \sum_{i=\alpha_x+1}^{m} t_i + \delta_x \;=\; u'_m(x) + \delta_x = u(x) + \delta_x$$

Hence $v'_m = v$. It follows that $\mathsf{first}(\pi') = u$ and $\mathsf{last}(\pi') = v$ as required. ■

6.6.2.2 Drifting clocks: $J^* \subseteq R_\varepsilon^*$.

We prove a result similar to Theorem 6.24 for drifting clocks.

Theorem 6.26 *Let A be a closed timed automaton, let $p = p_0\, p_1 \cdots p_N$ be a progress cycle of the bounded region automaton of A, and $\varepsilon \in \mathbb{R}^{>0}$. For all states $u, v \in L_p$, there exists a trajectory π of $[\![A]\!]_0^\varepsilon$ such that $\mathsf{first}(\pi) = u$ and $\mathsf{last}(\pi) = v$.*

The proof is more involved than for guard enlargement. We need the following intermediate lemmas.

Lemma 6.27 *Let A be a closed timed automaton, let r and r' be two regions of A such that $(\ell, r) \xrightarrow{\mathsf{time}} (\ell, r')$ in the bounded region automaton of A. For all $u \in r$, $v \in r'$ and $\tau \in \mathbb{R}^{>0}$ such that $(\ell, u) \xrightarrow{\tau} (\ell, v)$ in $[\![A]\!]_0^0$, and for all $\delta \geq 0$, for all $y \in \mathcal{N}_\infty(v, \delta) \cap [r']$, there exists $x \in \mathcal{N}_\infty(u, 2\delta) \cap [r]$ and $\tau' \in \mathbb{R}^{\geq 0}$ such that $(\ell, x) \xrightarrow{\tau'} (\ell, y)$ in $[\![A]\!]_0^0$.*

Proof. Let n be the number of clock of A. Reminiscent of the normal form DBM representation of regions, let $\alpha_i, \beta_i, m_{i,j} \in \mathbb{Z}$ and $\alpha'_i, \beta'_i, m'_{i,j} \in \mathbb{Z}$ be the tightest constants such that for all valuations u, v, we have $u \in [r]$ and $v \in [r']$ if and only if for all $1 \leq i, j \leq n$:

$$u_i - u_j \leq m_{i,j} \qquad \alpha_i \leq u_i \leq \beta_i \qquad ([r])$$
$$v_i - v_j \leq m'_{i,j} \qquad \alpha'_i \leq v_i \leq \beta'_i \qquad ([r'])$$

In particular, this entails that $-m_{j,i} \leq u_i - u_j \leq \beta_i - \alpha_j$ and $-m'_{j,i} \leq v_i - v_j \leq \beta'_i - \alpha'_j$ for all $1 \leq i, j \leq n$, and since the constants are tight:

$$-m_{j,i} \leq m_{i,j} \leq \beta_i - \alpha_j \qquad -m'_{j,i} \leq m'_{i,j} \leq \beta'_i - \alpha'_j \tag{6.4}$$

Now, let $u \in r$, $v \in r'$ and $\tau \in \mathbb{R}^{>0}$ such that $(\ell, u) \xrightarrow{\tau} (\ell, v)$ in $[\![A]\!]_0^0$, and let $\delta \geq 0$ and $y \in \mathcal{N}_\infty(v, \delta) \cap [r']$. Since r' is a time successor of r and $v = u + \tau$, we have for all $1 \leq i, j \leq n$:

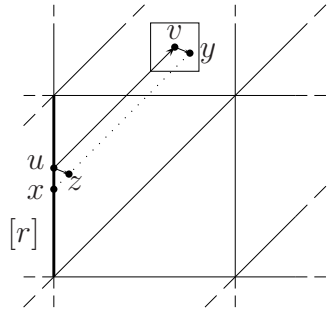
$$m_{i,j} = m'_{i,j} \qquad v_i - v_j = u_i - u_j \tag{6.5}$$

We define the valuation $D = y - v$. Since $y \in \mathcal{N}_\infty(v, \delta)$, we have $\|D\|_\infty \leq \delta$ and since $y \in [r']$, we have for all $1 \leq i, j \leq n$:

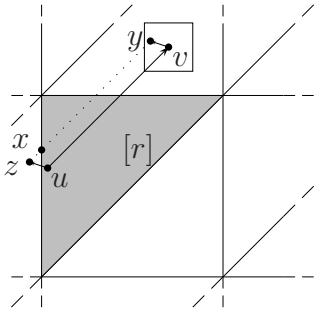
$$(v_i + D_i) - (v_j + D_j) \leq m_{i,j} \qquad \alpha'_i \leq v_i + D_i \leq \beta'_i \tag{6.6}$$

Now let $z = u + D$. As shown on Figure 6.5, we might have $z \notin [r]$. Thus, we have to construct a neighbour x of z that belongs to $[r]$ and such that $x \xrightarrow{\tau'} y$ for some $\tau' \in \mathbb{R}^{\geq 0}$. By Equation (6.5) and (6.6), we have for all $1 \leq i, j \leq n$:

$$z_i - z_j = (u_i + D_i) - (u_j + D_j) = (v_i + D_i) - (v_j + D_j) \leq m_{i,j} \tag{6.7}$$



(a) The interior of $[r]$ is empty.



(b) The interior of $[r]$ is not empty.

Figure 6.5: Construction of a predecessor of y in the closed region $[r]$.

1. First, assume that for some i_0, we have $\alpha_{i_0} = \beta_{i_0}$. This means that the interior of r is empty, as on Figure 6.5(a). Let $t = -D_{i_0}$. Notice that the value of t is independent of the choice of i_0. Indeed, if for some $j \neq i_0$ we have $\alpha_j = \beta_j$, then using Equation (6.4) we get:

$$\alpha_{i_0} - \beta_j \leq -m_{j,i_0} \leq m_{i_0,j} \leq \beta_{i_0} - \alpha_j$$

and $-m_{j,i_0} = m_{i_0,j}$ since $\alpha_{i_0} - \beta_j = \beta_{i_0} - \alpha_j$. By Equation (6.5), we have $-m'_{j,i_0} = m'_{i_0,j}$ and thus in the region $[r']$, we have $v_{i_0} - v_j = -m'_{j,i_0} = m'_{i_0,j} = y_{i_0} - y_j$ and thus $D_{i_0} = D_j$.

Now, let $x = z + t$ so that $x_{i_0} = u_{i_0}$. We show that $x \in [r]$. Clearly, by Equation (6.7) we have:

$$x_i - x_j = z_i - z_j \leq m_{i,j}$$

And in particular, for all $1 \leq j \leq n : -m_{i_0,j} \leq x_j - x_{i_0} \leq m_{j,i_0}$. Since $x_{i_0} = u_{i_0} = \alpha_{i_0} = \beta_{i_0}$ and by Equation (6.4) we have:

$$\alpha_j \leq \beta_{i_0} - m_{i_0,j} \leq x_j \leq m_{j,i_0} + \alpha_{i_0} \leq \beta_j$$

Now, we have $\|x - u\|_\infty = \|D + t\|_\infty \leq 2\|D\|_\infty \leq 2\delta$ and thus $x \in \mathcal{N}_\infty(u, 2\delta) \cap [r]$.

2. Second, assume that for all i, we have $\alpha_i < \beta_i$. This means that the interior of r is not empty, as on Figure 6.5(b). We define the following sets:

$$I = \{i \mid \alpha_i > z_i\} \qquad I' = \{i \mid z_i > \beta_i\}$$

- If $I = \varnothing$ and $I' = \varnothing$, then $z \in [r]$ by Equation (6.7) and we take $x = z$. We have $\|x - u\|_\infty = \|z - u\|_\infty = \|D\|_\infty \leq \delta$.

- If $I \neq \varnothing$, then define $t = \max\{\alpha_i - z_i \mid i \in I\}$ and let i_0 be an index in I such that $t = \alpha_{i_0} - z_{i_0}$. Clearly $t > 0$ and since $t = \alpha_{i_0} - u_{i_0} - D_{i_0}$ and $\alpha_{i_0} \leq u_{i_0}$, we have $t \leq -D_{i_0}$. We take $x = z + t$ so that $x_{i_0} = \alpha_{i_0}$. We show that $x \in [r]$. Clearly, by Equation (6.7) we have:

  $$x_i - x_j = z_i - z_j \leq m_{i,j}$$

  In particular, for all $1 \leq i \leq n$ we have $x_j - x_{i_0} \leq m_{j,i_0} \leq \beta_j - \alpha_{i_0}$ by Equation (6.4). Since $x_{i_0} = \alpha_{i_0}$, this yields $x_j \leq \beta_j$. Moreover, for all $i \in I$ we have $x_i = z_i + t \geq \alpha_i$ by definition of t, and for all $i \notin I$ we have $x_i = z_i + t \geq z_i \geq \alpha_i$. Finally, we have $\|x - u\|_\infty = \|D + t\|_\infty \leq 2\|D\|_\infty \leq 2\delta$.

- If $I' \neq \varnothing$, then define $t = \min\{\beta_i - z_i \mid i \in I'\}$ and let i_0 be an index in I' such that $t = \beta_{i_0} - z_{i_0}$. Clearly $t < 0$ and since $t = \beta_{i_0} - u_{i_0} - D_{i_0}$ and $u_{i_0} \leq \beta_{i_0}$, we have $t \geq -D_{i_0}$. We take $x = z + t$ so that $x_{i_0} = \beta_{i_0}$. We show that $x \in [r]$. Clearly, by Equation (6.7) we have:

  $$x_i - x_j = z_i - z_j \leq m_{i,j}$$

  In particular, for all $1 \leq j \leq n$ we have $x_{i_0} - x_j \leq m_{i_0,j} \leq \beta_{i_0} - \alpha_j$ by Equation (6.4). Since $x_{i_0} = \beta_{i_0}$, this yields $x_j \geq \alpha_j$. Moreover, for all $i \in I'$ we have $x_i = z_i + t \leq \beta_i$ by definition of t, and for all $i \notin I'$ we have $x_i = z_i + t \leq z_i \leq \beta_i$. Finally, we have $\|x - u\|_\infty = \|D + t\|_\infty \leq 2\|D\|_\infty \leq 2\delta$.

In each case, we have $x \in \mathcal{N}_\infty(u, 2\delta) \cap [r]$ and $(\ell, x) \xrightarrow{\tau'} (\ell, y)$ in $[\![A]\!]_0^0$ for $\tau' = \tau - t$ (obviously we have $\tau' \geq 0$ because r' is a time successor of r). ■

Now, intuitively, if we take a trajectory π from u to itself in the classical semantics $[\![A]\!]_0^0$, in order to reach the states in the neighbourhood of u (from u) in $[\![A]\!]_0^\varepsilon$, we can use the drifts to modify the timed transitions of π. To do so, we need to ensure that the duration of π is sufficiently long, because a fixed drift applies proportionately to durations. We care of that in the next lemma, by showing that if time passes on π then it is possible to cycle on u by a trajectory of duration at least $\frac{1}{2}$ without extending the size of π more than twice. This last condition is important as otherwise, the lemma would be trivially true.

Lemma 6.28 *Let A be a closed timed automaton, let $p = p_0\, p_1\, p_2 \ldots p_N$ be a progress cycle in the bounded region automaton of A. If there exists a limit cycle π of $[\![A]\!]$ that follows p such that $\mathsf{Duration}(\pi) > 0$, then there exists a limit cycle π' of $[\![A]\!]$ with $\mathsf{first}(\pi') = \mathsf{first}(\pi)$ and such that $|\pi'| \leq 2|\pi|$ and $\mathsf{Duration}(\pi') \geq 1/2$.*

Proof. The result is immediate if $\mathsf{Duration}(\pi) \geq 1/2$. Otherwise, assume that $\mathsf{Duration}(\pi) < 1/2$. Since all clocks are reset along π, their value is strictly less than $1/2$ in every state of the trajectory.

Let $k = |\pi|$ and $u = \mathsf{first}(\pi) = \mathsf{last}(\pi)$. Let π_2 be the trajectory obtained by repeating π twice. We have $|\pi_2| = 2k$ and $\mathsf{first}(\pi_2) = \mathsf{state}_k(\pi_2) = \mathsf{last}(\pi_2) = u$. Since all clocks are reset along π, their value remain strictly less than $1/2$ in every state of π and π_2. By the fact that $\mathsf{Duration}(\pi) > 0$, there must be at least one timed transition in π with a strictly positive time stamp. Consider the first such transition in π_2, and let increase its length by $1/2$ time units, yielding a new trajectory π' in which each clock remain below 1. Therefore, the same transitions as in π_2 can be taken as the guards satisfied by a state of π is also satisfied by the corresponding state in π'. Observe that we leave the second half of the trajectory π' unchanged, and since all clocks are reset along π, we obtain $\mathsf{last}(\pi') = u = \mathsf{first}(\pi')$, $|\pi'| = 2|\pi|$ and $\mathsf{Duration}(\pi') \geq 1/2$. ■

Lemma 6.29 *Let A be a closed timed automaton. Let (ℓ, x) and (ℓ, y) be two states of $[\![A]\!]$, and $\tau \in \mathbb{R}^{\geq 0}$ such that $(\ell, x) \xrightarrow{\tau} (\ell, y)$ in $[\![A]\!]$. For all $\varepsilon \in \mathbb{R}^{>0}$, for all $x' \in \mathcal{N}_\infty(x, \varepsilon\tau) : (\ell, x') \xrightarrow{\tau} (\ell, y)$ in $[\![A]\!]^\varepsilon_0$.*

Proof. The result is immediate if $\tau = 0$. Otherwise, it suffices to set the rate of each clock c of A to $1 - \frac{x'(c)-x(c)}{\tau}$, which lies between $1-\varepsilon$ and $1+\varepsilon$. ■

Lemma 6.30 *Let A be a closed timed automaton, let r and r' be two regions of A such that $r \rightarrow r'$ in the bounded region automaton of A. For all $u \in [r]$, $v \in [r']$ and $\varepsilon \in \mathbb{R}^{>0}$:*

- *if there is a timed transition $u \xrightarrow{\tau} v$ in $[\![A]\!]$, then for all $\eta \in \mathbb{R}^{>0}$, we have:*

$$\forall y \in \mathcal{N}_\infty(v, \frac{\eta + \varepsilon\tau}{2+3\varepsilon}) \cap [r'] \cdot \ \exists x' \in \mathcal{N}_\infty(u, \eta) \cap [r] : \ x' \xrightarrow{\tau'} y \text{ in } [\![A]\!]^\varepsilon_0;$$

- *if there is an action transition $u \xrightarrow{\sigma} v$ in $[\![A]\!]$, then for all $\eta \in \mathbb{R}^{>0}$, we have*

$$\forall y \in \mathcal{N}_\infty(v, \eta) \cap [r'] \cdot \ \exists x \in \mathcal{N}_\infty(u, \eta) \cap [r] : \ x \xrightarrow{\sigma} y \text{ in } [\![A]\!]^\varepsilon_0.$$

Proof. We only prove the first part of the lemma, the second part being quite obvious. We have $v = u + \tau$. Let $\delta = K_\varepsilon(\eta + \varepsilon\tau)$ and let $y \in \mathcal{N}_\infty(v, \delta) \cap [r']$. From Lemma 6.27, there exists $x \in \mathcal{N}_\infty(u, 2\delta) \cap [r]$ such that $x \xrightarrow{\tau'} y$ in $[\![A]\!]$ for some $\tau' \in \mathbb{R}^{\geq 0}$. So we have $y = x + \tau'$. Using the triangle inequalities, we have:

$$\tau' = \|y - x\|_\infty = \|(v-u) - [(x-u) + (v-y)]\|_\infty \geq \tau - 3\delta \tag{6.8}$$

Consider the set $S = \mathcal{N}_\infty(x, \varepsilon\tau') \cap \mathsf{Conv}(\{u, x\})$. Since $d_\infty(u, x) \leq 2\delta$, there exists $x' \in S$ such that:

$$\begin{cases} d_\infty(u, x') = 0 & \text{if } d_\infty(u, x) \leq \varepsilon\tau' \text{ (take } x' = u) \\ d_\infty(u, x') \leq 2\delta - \varepsilon\tau' & \text{if } d_\infty(u, x) > \varepsilon\tau' \end{cases}$$

Since $[r]$ is convex and $x, u \in [r]$, we have $x' \in [r]$, and since $x \xrightarrow{\tau'} y$ in $[\![A]\!]$, Lemma 6.29 entails that $x' \xrightarrow{\tau'} y$ in $[\![A]\!]^{\varepsilon}_0$. To complete the proof, we have to show that $x' \in \mathcal{N}_\infty(u, \eta)$, that is $d_\infty(u, x') \leq \eta$. Starting from Equation (6.8), we have:

$$\varepsilon(\tau - \tau') \leq 3\varepsilon\delta = (2 + 3\varepsilon)\delta - 2\delta = \eta + \varepsilon\tau - 2\delta$$

and thus $2\delta - \varepsilon\tau' \leq \eta$ which entails $d_\infty(u, x') \leq \eta$. ■

Lemma 6.31 *Let A be a closed timed automaton, let $\varepsilon \in \mathbb{R}^{>0}$ and $K_\varepsilon = 1/(2 + 3\varepsilon)$. Let $p = p_0\, p_1 \cdots p_N$ be a path in the bounded region automaton of A. Let π be a trajectory of $[\![A]\!]$ that follows p and let $u = \mathsf{first}(\pi)$, $v = \mathsf{last}(\pi)$ and $T = \mathsf{Duration}(\pi)$. For all $y \in \mathcal{N}_\infty(v, K_\varepsilon^N \varepsilon T) \cap [p_N]$, there exists a trajectory π' in $[\![A]\!]^{\varepsilon}_0$ that follows p and such that $\mathsf{first}(\pi') = u$ and $\mathsf{last}(\pi') = y$.*

Proof. Let $\pi = (q_0, t_0)\sigma_1(q_1, t_1)\sigma_2 \ldots \sigma_N(q_N, t_N)$ with $t_0 = 0$ and $t_N = T$. Define $\epsilon_i = K_\varepsilon^i \varepsilon t_i$. We show that for all $0 \leq i < N$, for all $y \in \mathcal{N}_\infty(q_{i+1}, \epsilon_{i+1}) \cap [r_{i+1}]$, there exists $x \in \mathcal{N}_\infty(q_i, \epsilon_i) \cap [r_i]$ such that there exists a transition from x to y in $[\![A]\!]^{\varepsilon}_0$:

- if $q_i \xrightarrow{\sigma_{i+1}} q_{i+1}$ is a discrete transition, then we have $\epsilon_{i+1} \leq \epsilon_i$ because $t_{i+1} = t_i$ and $K_\varepsilon \leq 1$. The claims follows then directly from Lemma 6.30.
- otherwise, we have a timed transition $q_i \xrightarrow{\tau} q_{i+1}$ and $t_{i+1} = t_i + \tau$. By Lemma 6.30 with $\eta = \epsilon_i$, we have:

  $$\forall y \in \mathcal{N}_\infty(q_{i+1}, K_\varepsilon(\epsilon_i + \varepsilon\tau)) \cap [r_{i+1}] \cdot \exists x' \in \mathcal{N}_\infty(q_i, \epsilon_i) \cap [r_i] :\ x' \xrightarrow{\tau'} y \text{ in } [\![A]\!]^{\varepsilon}_0.$$

  Since $K_\varepsilon \leq 1$, we have:

  $$\begin{aligned} K_\varepsilon(\epsilon_i + \varepsilon\tau) &= K_\varepsilon^{i+1}\varepsilon t_i + K_\varepsilon \varepsilon\tau \\ &\geq K_\varepsilon^{i+1}\varepsilon(t_i + \tau) \\ &= K_\varepsilon^{i+1}\varepsilon t_{i+1} = \epsilon_{i+1} \end{aligned}$$

  Hence, $\mathcal{N}_\infty(q_{i+1}, \epsilon_{i+1}) \subseteq \mathcal{N}_\infty(q_{i+1}, K_\varepsilon(\epsilon_i + \varepsilon\tau))$ and we have:

  $$\forall y \in \mathcal{N}_\infty(q_{i+1}, \epsilon_{i+1}) \cap [r_{i+1}].\ \exists x' \in \mathcal{N}_\infty(q_i, \epsilon_i) \cap [r_i].\ x' \xrightarrow{\tau'} y \text{ in } [\![A]\!]^{\varepsilon}_0.$$

Applying this result for each $0 \leq i < N$, we obtain immediately that for all $y \in \mathcal{N}_\infty(q_N, \epsilon_N) \cap [r_N]$, there exists $x \in \mathcal{N}_\infty(q_0, \epsilon_0) \cap [r_0]$ such that there exists a trajectory π' in $[\![A]\!]^{\varepsilon}_0$ that follows p with $\mathsf{first}(\pi') = x$ and $\mathsf{last}(\pi') = y$. Finally, we have $q_N = \mathsf{last}(\pi)$ and $q_0 = \mathsf{first}(\pi)$ so that $x = u$ since $\epsilon_0 = 0$ and $\mathcal{N}_\infty(q_0, 0) = \{q_0\}$. ■

Lemma 6.32 *Let A be a closed timed automaton and p be a progress cycle of the bounded region automaton of A. For all $u, v \in L_p$, there exists an $n \in \mathbb{N}$ such that $\mathsf{Conv}(\{u, v\}) \subseteq L_{n,p}$.*

Proof. Let k and l be such that $u \in L_{k,p}$ and $v \in L_{l,p}$. Take $n = kl$. The result follows from Lemma 6.20. ■

Lemma 6.33 *Let A be a closed timed automaton and $p = p_0\, p_1 \cdots p_N$ be a progress cycle of the bounded region automaton of A. For all $u, v \in L_p$, for all $\varepsilon \in \mathbb{R}^{>0}$, there exists $\delta > 0$ such that for all $x \in \mathsf{Conv}(\{u, v\})$ and for all $y \in L_p \cap \mathcal{N}_\infty(x, \delta)$, there exists a trajectory π in $[\![A]\!]^\varepsilon_0$ such that $\mathsf{first}(\pi) = x$ and $\mathsf{last}(\pi) = y$.*

Proof. If p is not a time-elapsing progress cycle, then L_p is a singleton that contains the valuation in which all clocks are equal to zero. In this case, the result is immediate.

Assume that p contains a time-elapsing region. For $u, v \in L_p$, let $n \in \mathbb{N}$ be given by Lemma 6.32. We are in the conditions of Lemma 6.28: for all $x \in \mathsf{Conv}(\{u, v\})$ there exists a limit cycle π on x with $\mathsf{Duration}(\pi) > 0$ and $|\pi| \leq nW$ where W is the number of regions of A. Therefore, there exists a limit cycle π' on x with $\mathsf{Duration}(\pi') \geq 1/2$ and $|\pi'| \leq 2nW$. Let $N = 2nW$ and take $\delta = \frac{1}{2}\varepsilon K_\varepsilon^N$. By Lemma 6.31, for all $y \in \mathcal{N}_\infty(x, \delta) \cap [p_0]$ there exists a trajectory π in $[\![A]\!]^\varepsilon_0$ such that $\mathsf{first}(\pi) = x$ and $\mathsf{last}(\pi) = y$. Finally, the result follows from the fact that $L_p \subseteq [p_0]$. ■

We proceed with the proof of Theorem 6.26:

Proof of Theorem 6.26. For $u, v \in L_p$, let δ as given by Lemma 6.33 and let $k = \lceil \frac{1}{\delta} \rceil$. Consider the points $x_0 = u$, $x_k = v$, and $x_i = u + i\delta(v - u)$ for $i = 1, \ldots, k-1$. It is easy to see that $d_\infty(x_i, x_{i+1}) \leq \delta \cdot d_\infty(u, v) \leq \delta$ (because the ∞-distance between two points of a region is at most 1). Thus from Lemma 6.33, for all $0 \leq i \leq k-1$ there exists a trajectory from x_i to x_{i+1} in $[\![A]\!]^\varepsilon_0$, and thus a trajectory π such that $\mathsf{first}(\pi) = u$ and $\mathsf{last}(\pi) = v$. ■

6.6.2.3 Soundness of Algorithm 4.

Theorem 6.34 *Let A be a closed timed automaton. Let $p = p_0\, p_1 \ldots p_N$ be a progress cycle of the bounded region automaton of A. For all $x, y \in [p_0]$, we have:*

- *For all $\Delta \in \mathbb{R}^{>0}$, there exists a trajectory π in $[\![A]\!]^0_\Delta$,*
- *For all $\varepsilon \in \mathbb{R}^{>0}$, there exists a trajectory π' in $[\![A]\!]^\varepsilon_0$*

such that $\mathsf{first}(\pi) = \mathsf{first}(\pi') = x$ *and* $\mathsf{last}(\pi) = \mathsf{last}(\pi') = y$.

Proof. From Theorem 6.19, there exist $u, v \in L_p$ and two trajectories π_1 and π_3 of $[\![A]\!]$ such that $\mathsf{first}(\pi_1) = x$ and $\mathsf{last}(\pi_1) = u$, and $\mathsf{first}(\pi_3) = v$ and $\mathsf{last}(\pi_3) = y$. By Theorem 6.24, there exists a trajectory π_2 of $[\![A]\!]^0_\Delta$ such that $\mathsf{first}(\pi_2) = u$ and $\mathsf{last}(\pi_2) = v$. We construct π by concatenating the three trajectories π_1, π_2 and π_3. The proof is similar for the second part of the theorem, based on Theorem 6.26. ■

As a consequence:

Theorem 6.35 *Let J^* be the set computed by Algorithm 4. We have $J^* \subseteq R^*_\Delta$ and $J^* \subseteq R^*_\varepsilon$.*

Proof. For all $\Delta > 0$, if a set of regions J^* is reachable in $[\![A]\!]^0_\Delta$, then:

- so is the set $\mathsf{Reach}(R_A, J^*)$ of regions reachable from J^* in the bounded region automaton R_A of A;
- for all progress cycles p, if p_0 is a region in p such that $[p_0] \cap J^* \neq \varnothing$, then the set $J^* \cup [p_0]$ is reachable in $[\![A]\!]^0_\Delta$, according to Theorem 6.34.

Since J^* is obtained by iterating the above two operations (lines 4, 6 and 7 of the algorithm) from the set of initial states Q_0 (see line 3), this ensures that $J^* \subseteq \mathsf{Reach}([\![A]\!]^0_\Delta)$. This holds for all $\Delta > 0$, and hence $J^* \subseteq R^*_\Delta$.

The proof for drifts on clocks is similar. ■

6.6.3 Completeness of Algorithm 4: $R^*_{\Delta,\varepsilon} \subseteq J^*$

To prove the completeness of Algorithm 4, we have to show that any state that is reachable in the semantics $[\![A]\!]^\varepsilon_\Delta$ no matter how small are ε and Δ, lies in the set J^* computed by Algorithm 4. First, we show that if the number of transitions in trajectories is *bounded*, we can compute a bound on the distance between a state reachable in $[\![A]\!]^\varepsilon_\Delta$ and the set of reachable states $\mathsf{Reach}([\![A]\!])$ in the classical semantics (Theorem 6.40). This bound vanishes when $\varepsilon \to 0$ and $\Delta \to 0$. This shows that a state $x \in R^*_{\Delta,\varepsilon}$ that is reachable in a bounded number of steps from the initial states in $[\![A]\!]^\varepsilon_\Delta$ for all $\varepsilon, \Delta > 0$ is at distance zero from the reachable states in the classical semantics. An argument related to topologically closed sets then shows that $x \in \mathsf{Reach}([\![A]\!])$. Second, to extend the result to the whole set $R^*_{\Delta,\varepsilon}$, we roughly use the fact that longer trajectories necessarily contain a cycle that is added to J^* by the algorithm. Therefore, only a bounded part of those trajectories can take the state far from J^*. By a similar argument as above, this distance to J^* is shown to vanish when $\varepsilon \to 0$ and

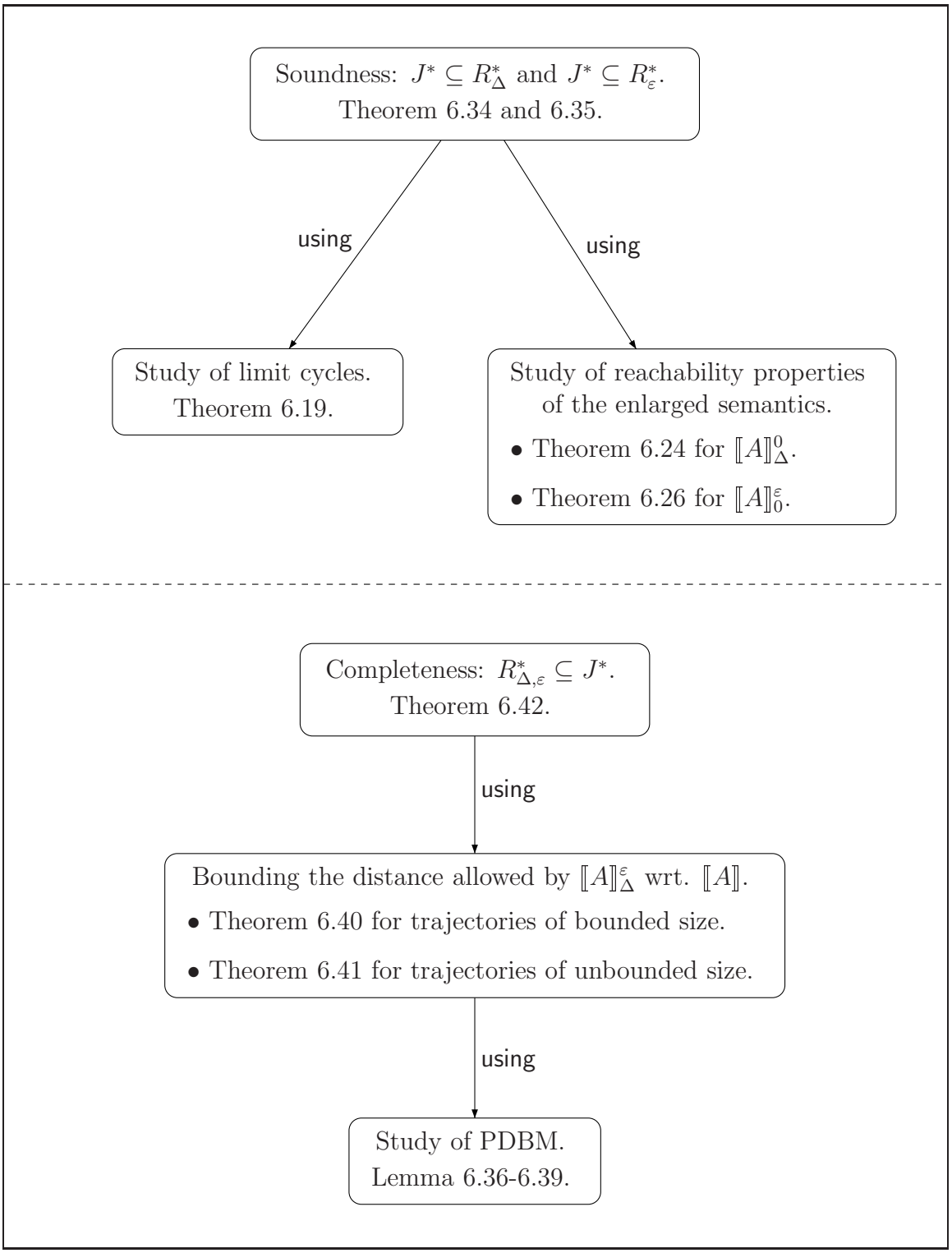


Figure 6.6: Milestones for the proofs of soundness and completeness of Algorithm 4.

$\Delta \to 0$ (Theorem 6.41). The proofs of the theorems are based on a detailed study of the reachability properties of the enlarged semantics, for which we use a tool that may appear very powerful for our purpose: the parametric DBM, an extension of DBM that we have presented in Section 6.4). Our attempts to come up with simpler or more intuitive proofs have failed. The important steps of the proof can be followed on Figure 6.6.

A *parametric* DBM (PDBM) in $\mathbb{R}^n$ is a matrix $\mathbf{M} = \big(m_{i,j}\big)_{0\le i,j\le n}$ where $m_{i,j} \in \mathbb{Z}\times\mathbb{N}$ is called a *parametric bound*. In a PDBM, each $m_{i,j}$ is a couple (a, b) of integers with $b \ge 0$. Given a number $\Omega \in \mathbb{R}^{\ge 0}$, the *value* of $m_{i,j}$ is $[\![m]\!]_\Omega = a + b\Omega$. The set represented by $\mathbf{M}$ is:

$$[\![\mathbf{M}]\!]_\Omega = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid \forall\, 0 \le i, j \le n :\ x_i - x_j \le [\![m_{ij}]\!]_\Omega \wedge\ x_0 = 0\}.$$

As usual, we often write $[\![\mathbf{M}]\!]$ for $[\![\mathbf{M}]\!]_0$. More general definitions of PDBM have been introduced in [AAB00, HRSV01], with implementations. Here, we use PDBM for purely theoretical purposes, so we keep the definition as simple as possible.

For a PDBM $\mathbf{M} = \big(m_{ij}\big)_{0\le i,j\le n}$ with $m_{ij} = (a_{ij}, b_{ij})$, we define the *width* of $\mathbf{M}$ by $w(\mathbf{M}) = \max\{b_{ij} \mid 0 \le i, j \le n\}$. Thus a DBM is a zero-width PDBM. Any closed rectangular guard g can be represented by a PDBM $\mathbf{M}_g$ with $w(\mathbf{M}_g) = 2$ such that for all $\Omega \in \mathbb{R}^{\ge 0}$ we have $[\![\mathbf{M}_g]\!]_\Omega = \mathcal{N}_\infty([\![g]\!], \Omega)$. In particular, $[\![g]\!] = [\![\mathbf{M}_g]\!]$.

Example Let $g \equiv x = 4 \wedge 1 \le y \le 3$. Then,

$$\mathbf{M}_g = \begin{array}{c} \\ \mathsf{0} \\ \mathsf{x} \\ \mathsf{y} \end{array} \begin{array}{c} \begin{array}{ccc} \mathsf{0} & \mathsf{x} & \mathsf{y} \end{array} \\ \begin{pmatrix} (0,0) & (-4,1) & (-1,1) \\ (4,1) & (0,0) & (3,2) \\ (3,1) & (-1,2) & (0,0) \end{pmatrix} \end{array}$$

When the reachable states in the enlarged semantics of a closed timed automaton A are computed parametrically using PDBM, it would be nice that the classical semantics $[\![\mathbf{M}]\!]$ gives exactly the reachable states in $[\![A]\!]$ and that the enlarged semantics $[\![\mathbf{M}]\!]_\Omega$ gives the reachable states in $[\![A]\!]^\varepsilon_\Delta$. This can be obtained when $\varepsilon = 0$ by taking $\Omega = \Delta$. For the general case $\varepsilon > 0$, the set $[\![\mathbf{M}]\!]_\Omega$ over-approximates the reachable states, provided ε is sufficiently small. We are more precise in Lemma 6.38 and Lemma 6.39.

In that context, the width of PDBM records the accumulation of the deviations allowed by the enlarged semantics. This is useful to bound the distance between states that are reachable in the enlarged semantics and states that are reachable in the classical semantics. The following lemma gives such a bound.

Lemma 6.36 *Let $\mathbf{M}$ be a PDBM in $\mathbb{R}^n$ and let $\Omega \in \mathbb{R}^{\ge 0}$ such that $\Omega\cdot(2n+1)\cdot w(\mathbf{M}) < 1$. Let $Z = [\![\mathbf{M}]\!]$ and $Z' = [\![\mathbf{M}]\!]_\Omega$. For all $x' \in Z'$, there exists $x \in Z$ such that $\|x' - x\|_\infty \le n \cdot w(\mathbf{M}) \cdot \Omega$.*

Proof. First, assume that x' is a vertex of Z'. Then x' can be obtained by solving a system of n equations of the form $x'_i - x'_j = [\![m_{ij}]\!]_\Omega$, $x'_i = [\![m_{i0}]\!]_\Omega$ or $x'_i = -[\![m_{0i}]\!]_\Omega$. Therefore, each x'_i is the sum or difference of at most n coefficients $[\![m_{ij}]\!]_\Omega$. Since the bounds m_{ij} are entries of $\mathbf{M}$, for all $1 \leq i \leq n$, if $x'_i = l_i + k_i\Omega$ for some $l_i, k_i \in \mathbb{Z}$, then $|k_i| \leq n \cdot w(\mathbf{M})$ and we take $x_i = l_i$. Then $\|x' - x\|_\infty \leq n \cdot w(\mathbf{M}) \cdot \Omega$ and we claim that $x \in Z$. Let $l_0 = k_0 = 0$. Then, for all $0 \leq i, j \leq n$ we have:

$$x'_i - x'_j = l_i - l_j + (k_i - k_j) \cdot \Omega \leq a_{ij} + b_{ij}\Omega$$

Hence,

$$l_i - l_j \leq a_{ij} + (b_{ij} - k_i + k_j) \cdot \Omega$$

Since l_i, l_j and a_{ij} are integers and $|(b_{ij} - k_i + k_j) \cdot \Omega| \leq (2n+1) \cdot w(\mathbf{M}) \cdot \Omega < 1$, we have $x_i - x_j = l_i - l_j \leq a_{ij}$. Therefore $x \in Z$.

Second, if x' is not a vertex, then it can be written as $x' = \sum_i \lambda_i v'_i$ with $\lambda_i \geq 0$ and $\sum_i \lambda_i = 1$ and each v'_i is a vertex of Z'. From the proof above, for each v'_i there exists $v_i \in Z$ such that $\|v'_i - v_i\|_\infty \leq n \cdot w(\mathbf{M}) \cdot \Omega$. We take $x = \sum_i \lambda_i v_i$. Clearly $x \in Z$, and we have:

$$\begin{aligned}
\|x' - x\|_\infty &= \Big\| \sum_i \lambda_i (v'_i - v_i) \Big\|_\infty \\
&\leq \sum_i \lambda_i \|v'_i - v_i\|_\infty \\
&\leq \sum_i \lambda_i (n \cdot w(\mathbf{M}) \cdot \Omega) \\
&\leq n \cdot w(\mathbf{M}) \cdot \Omega
\end{aligned}$$

■

Now, we show how to extend to PDBM the operations that we have presented in Section 6.4 for DBM. To do so, we have to define the minimum of two parametric bounds (for intersection of PDBM). We define a lexicographic order on parametric bounds: $(a, b) \leq (a', b')$ if and only of either $a < a'$, or $a = a'$ and $b \leq b'$. This (syntactical) definition is justified by the following observation: for all Ω such that $b\Omega \leq 1$, if $(a, b) \leq (a', b')$ then $[\![(a, b)]\!]_\Omega \leq [\![(a', b')]\!]_\Omega$. Thus if we take a sufficiently small Ω, the order is preserved at the semantical level. In the sequel, this will imply that provided Ω is below some threshold, the operations on PDBM can be performed *independently* of the value of Ω. The *sum* of two parametric bounds (a, b) and (a', b') is $(a + a', b + b')$.

We review the fundamental operations on PDBM:

- Intersection: the intersection of two PDBM $\mathbf{M}_1$ and $\mathbf{M}_2$ is the PDBM $\mathbf{M}$ whose entries are the minimum (according to the lexicographic order on parametric bounds) of the corresponding entries of $\mathbf{M}_1$ and $\mathbf{M}_2$. Hence $w(\mathbf{M}) \leq w(\mathbf{M}_1)$ and $w(\mathbf{M}) \leq w(\mathbf{M}_2)$.

- Time passing and reset: those operations only substitute entries of the matrix with other entries of the matrix and they preserve the normal form (see below). Thus the width cannot increase.

- Normalization: to obtain the normal form of a PDBM $\mathbf{M}$ in $\mathbb{R}^n$, each entry m_{ij} is replaced by the length of the shortest path from node i to node j, which has at most n edges. Therefore, the width of the normal form is bounded by $n \cdot w(\mathbf{M})$.

- Emptiness test: given a PDBM $\mathbf{M}$, let $\mathbf{M}'$ be its normal form. The emptiness test checks whether one of the diagonal entries is negative (a parametric bound $m = (a, b)$ is *negative* iff $m < (0, 0)$ iff $a \leq -1$).

A summary of the above observations is given in Table 6.1. Note that all the operations (except the normalization) are correct for all values of Ω. For all PDBM $\mathbf{M}$, $\mathbf{M}'$, $\mathbf{M}_1$ and $\mathbf{M}_2$ in $\mathbb{R}^n$, we have:

- $\forall\, \Omega \in \mathbb{R}^{\geq 0} : [\![\mathbf{M_1} \cap \mathbf{M_2}]\!]_\Omega = [\![\mathbf{M_1}]\!]_\Omega \cap [\![\mathbf{M_2}]\!]_\Omega$;
- $\forall\, \Omega \in \mathbb{R}^{\geq 0} : [\![\mathbf{M}\nearrow]\!]_\Omega = [\![\mathbf{M}]\!]_\Omega\nearrow$;
- $\forall\, \Omega \in \mathbb{R}^{\geq 0} \cdot \forall R \subseteq \{x_1, \ldots, x_n\} : [\![\mathbf{M}[R := 0]]\!]_\Omega = [\![\mathbf{M}]\!]_\Omega[R := 0]$;
- $\forall\, \Omega \in [0, 1/(n \cdot w(\mathbf{M}))]$: if $\mathbf{M}'$ is the normal form of $\mathbf{M}$, then the DBM $\big([\![m'_{ij}]\!]_\Omega\big)_{0 \leq i,j \leq n}$ is the normal form of the DBM $\big([\![m_{ij}]\!]_\Omega\big)_{0 \leq i,j \leq n}$.

For the emptiness test, the value of Ω should also be bounded.

Lemma 6.37 *Let* $\mathbf{M}$ *be a PDBM. We have:*

$$\forall\, \Omega \in [0, 1/(n \cdot w(\mathbf{M}))[:\ [\![\mathbf{M}]\!]_\Omega = \varnothing \text{ iff } [\![\mathbf{M}]\!]_0 = \varnothing$$

Proof. First, we have $[\![\mathbf{M}]\!]_0 \subseteq [\![\mathbf{M}]\!]_\Omega$ for all Ω. Thus it suffices to show that $[\![\mathbf{M}]\!]_0 = \varnothing$ implies that $[\![\mathbf{M}]\!]_\Omega = \varnothing$ for all $\Omega < 1/(n \cdot w(\mathbf{M}))$. If $[\![\mathbf{M}]\!]_0 = \varnothing$ then there exists a parametric bound $m' = (a, b)$ in the diagonal of the normal form PDBM $\mathbf{M}'$ such that $a \leq -1$. Since $b \leq n \cdot w(\mathbf{M})$, we have $[\![m']\!]_\Omega = a + b\Omega < 0$ and therefore $[\![\mathbf{M}]\!]_\Omega$ is empty. ■

Notation Given a TTS $\mathcal{T} = \langle Q, Q_0, Q_f, \Sigma, \rightarrow \rangle$, let $A \subseteq Q$ and $\sigma \in \Sigma$. We define the following operators:

$$\mathsf{post}^\sigma_{\mathcal{T}}(A) = \{q' \in Q \mid \exists q \in A :\ q \xrightarrow{\sigma} q'\}$$
$$\mathsf{post}^{\mathsf{time}}_{\mathcal{T}}(A) = \{q' \in Q \mid \exists q \in A \cdot \exists t \in \mathbb{R}^{\geq 0} :\ q \xrightarrow{t} q'\}$$

PDBM in $\mathbb{R}^n$	Input in NF	Output in NF	Width of the result
Intersection $\mathbf{M_1} \cap \mathbf{M_2}$	No	No	$\leq \max\{w(\mathbf{M_1}), w(\mathbf{M_2})\}$
Time passing $\mathbf{M}^{\nearrow}$	Yes	Yes	$\leq w(\mathbf{M})$
Reset $\mathbf{M}[R := 0]$	Yes	Yes	$\leq w(\mathbf{M})$
Normalization of $\mathbf{M}$ Emptiness test of $\mathbf{M}$	No	Yes	$\leq n \cdot w(\mathbf{M})$

Table 6.1: Operations on PDBM (NF = normal form).

We use the PDBM to characterize the relationship between the reachable states of the classical semantics $[\![A]\!]$ and those of the enlarged semantics $[\![A]\!]^\varepsilon_\Delta$.

By an abuse of notation, we omit the location in the argument of $\mathsf{post}(\cdot)$, that is we use $Z = [\![\mathbf{M}]\!]$ instead of $Z = \{\ell\} \times [\![\mathbf{M}]\!]$ for $\ell \in \mathsf{Loc}$. Finally, we assume that the edges of timed automata are identified by their label. This is clearly not restrictive for reachability analysis.

In Lemma 6.38, the PDBM $\mathbf{M}'$ contains the exact information about the timed successors of $\mathbf{M}$ in the classical semantics, and it is an over-approximation of the timed successors in the enlarged semantics. Lemma 6.39 is similar for discrete successors.

Lemma 6.38 *Let A be a closed timed automaton with n clocks and largest constant M. Let $\mathbf{M}$ be a PDBM in $\mathbb{R}^n$ in normal form. There exists a PDBM $\mathbf{M}'$ in normal form such that:*

- $\forall\, \Omega \in \mathbb{R}^{\geq 0} \cdot \forall \Delta \in \mathbb{R}^{\geq 0} \cdot \forall \varepsilon \leq \Omega/(2(M+1)) : \mathsf{post}^{\mathsf{time}}_{[\![A]\!]^\varepsilon_\Delta}([\![\mathbf{M}]\!]_\Omega) \subseteq [\![\mathbf{M}']\!]_\Omega$;
- $\mathsf{post}^{\mathsf{time}}_{[\![A]\!]^0_0}([\![\mathbf{M}]\!]_0) = [\![\mathbf{M}']\!]_0$;
- $w(\mathbf{M}') = w(\mathbf{M}) + 1$.

Proof. Assume that $\Omega, \Delta \in \mathbb{R}^{\geq 0}$ and $\varepsilon \leq \Omega/(2(M+1))$. First, observe that in the classical semantics $[\![A]\!]$, the length of a timed transition is bounded by M. In the enlarged semantics $[\![A]\!]^\varepsilon_\Delta$ however, a timed transition may be longer than M because clocks can progress slower, namely at the rate $1-\varepsilon$. Therefore, the length of a timed transition is bounded by $M/(1-\varepsilon)$ and thus by $M+1$ since $\varepsilon \leq 1/(M+1)$. Second, we obtain $\mathbf{M}'$ by constructing the time successor of $\mathbf{M}$ as described above (in the exact semantics), and then by replacing each bound (a, b) of the PDBM by $(a, b+1)$, except on the diagonal. Clearly we have $w(\mathbf{M}') = w(\mathbf{M}) + 1$ and $[\![\mathbf{M}^{\nearrow}]\!]_0 = [\![\mathbf{M}']\!]_0$ and thus

$\mathsf{post}^{\mathsf{time}}_{[\![A]\!]^0_0}([\![\mathbf{M}]\!]_0) = [\![\mathbf{M}']\!]_0$. On the other hand, if we have $(\ell, x) \xrightarrow{t} (\ell, x')$ in $[\![A]\!]^\varepsilon_\Delta$ and $x_i - x_j \leq [\![m_{ij}]\!]_\Omega$, then:

$$x'_i - x'_j \leq [\![m_{ij}]\!]_\Omega + 2\varepsilon t \leq [\![m_{ij}]\!]_\Omega + 2\varepsilon(M+1) \leq [\![m_{ij}]\!]_\Omega + \Omega = [\![m'_{ij}]\!]_\Omega$$

Therefore $\mathsf{post}^{\mathsf{time}}_{[\![A]\!]^\varepsilon_\Delta}([\![\mathbf{M}]\!]_\Omega) \subseteq [\![\mathbf{M}']\!]_\Omega$. ■

Lemma 6.39 *Let A be a closed timed automaton with n clocks and alphabet Lab. Let $\mathbf{M}$ be a PDBM in $\mathbb{R}^n$. For all $\sigma \in \mathsf{Lab}$, there exists a PDBM $\mathbf{M}'$ in normal form such that:*

- $\forall\, \Omega \in \mathbb{R}^{\geq 0} \cdot \forall \Delta \leq \Omega \cdot \forall \varepsilon \in \mathbb{R}^{\geq 0} : \mathsf{post}^\sigma_{[\![A]\!]^\varepsilon_\Delta}([\![\mathbf{M}]\!]_\Omega) \subseteq [\![\mathbf{M}']\!]_\Omega$;
- $\mathsf{post}^\sigma_{[\![A]\!]^0_0}([\![\mathbf{M}]\!]_0) = [\![\mathbf{M}']\!]_0$;
- $w(\mathbf{M}') \leq n \cdot \max\{2, w(\mathbf{M})\}$.

Proof. Assume that $\Omega, \varepsilon \in \mathbb{R}^{\geq 0}$ and $\Delta \leq \Omega$. Let $(\ell, \ell', g, \sigma, R)$ be the edge of A associated to σ. Let $\mathbf{M}_g$ be the PDBM that represents the guard g. To construct $\mathbf{M}'$, let $\mathbf{M}_\cap$ be the PDBM $\mathbf{M} \cap \mathbf{M}_g$ put in normal form, and let $\mathbf{M}' = \mathbf{M}_\cap[R := 0]$ which is in normal form. According to Table 6.1, we have $w(\mathbf{M}') \leq n \cdot \max\{2, w(\mathbf{M})\}$ and

$$\mathsf{post}^\sigma_{[\![A]\!]^\varepsilon_\Delta}([\![\mathbf{M}]\!]_\Omega) \subseteq \mathsf{post}^\sigma_{[\![A]\!]^\varepsilon_\Omega}([\![\mathbf{M}]\!]_\Omega) = [\![\mathbf{M}']\!]_\Omega$$

For $\Omega = \Delta = \varepsilon = 0$, the sets collapse and $\mathsf{post}^\sigma_{[\![A]\!]^0_0}([\![\mathbf{M}]\!]_0) = [\![\mathbf{M}']\!]_0$. ■

With the two previous lemma, we have characterized how much the set of reachable states can increase by taking *one* transition (either timed or discrete) in the enlarged semantics $[\![A]\!]^\varepsilon_\Delta$ instead of the classical semantics $[\![A]\!]^0_0$. That increase is measured in terms of the width of a PDBM. In the next lemma, we use an argument by induction to give a bound on the increase after a given number of transitions. However, this is not sufficient to prove the completeness of Algorithm 4. We need in addition to show that every trajectory π' in $[\![A]\!]^\varepsilon_\Delta$ can be approached by a trajectory π in $[\![A]\!]^0_0$ where each intermediate state in π is "close" to the corresponding state in π'. To obtain this result, we introduce the notion of *automaton refinement* that roughly divides the guards into small pieces of size[4] $\frac{1}{\gamma}$ (with $\gamma \in \mathbb{N}$) so that two valuations that satisfy the same guard are necessarily "close" to each other (by choosing γ sufficiently large). This is the core of Theorem 6.40.

[4]The *size* of a set is the maximal ∞-distance between two points in the set.

Automaton refinement Given a closed timed automaton A with n clocks and an integer $\gamma \in \mathbb{N}$, the *γ-refinement* of A is the closed timed automaton A_γ constructed from A by first substituting in A each constant c appearing in the rectangular constraints (guards, invariants, initial and final conditions) of A by $c\gamma$ and second removing each edge (l, l', g, σ, R) of A and replacing it by the set of all edges (l, l', g', σ, R) such that $g' \in \phi(g)$ where $\phi(g)$ is defined as follows. Assume without loss of generality that $g \equiv \varphi_1 \wedge \cdots \wedge \varphi_n$ where for all $1 \leq i \leq n$ the constraint φ_i is of the form either $\varphi_i \equiv x_i = a_i$ or $\varphi_i \equiv a_i \leq x_i \leq b_i$ with $a_i, b_i \in \gamma\mathbb{N}$. Correspondingly, let ϕ_i be the set of constraints $\{x_i = a_i\}$ or $\{c \leq x_i \leq c+1 \mid c \in \mathbb{N} \wedge a_i \leq c < b_i\}$. Now, we define $\phi(g) = \{g' \equiv \varphi'_1 \wedge \cdots \wedge \varphi'_n \mid \forall\, 1 \leq i \leq n : \varphi'_i \in \phi_i\}$.

Roughly, the γ-refinement of A is a scaling of the constants by a factor γ (and thus a scaling of the time), followed by a partitioning of the guards such that the ∞-distance between two valuations that satisfy the guard is at most 1 (instead of a multiple of γ after the scaling).

The important property of such refinements is that for all $\Delta, \varepsilon \in \mathbb{R}^{\geq 0}$, the two TTS $[\![A]\!]^\varepsilon_\Delta$ and $[\![A_\gamma]\!]^\varepsilon_{\gamma\Delta}$ are bisimilar, witnessed by the bijection $\mu_\gamma \colon Q_A \to Q_{A_\gamma}$ such that $\mu_\gamma(\ell, v) = (\ell, \gamma v)$. We extend μ_γ to trajectories as expected (states are mapped according to μ_γ and the time stamps are multiplied by γ). Finally, for all $v, v' \in Q_{A_\gamma}$ we have $\|\mu_\gamma^{-1}(v) - \mu_\gamma^{-1}(v')\|_\infty = \|v - v'\|_\infty / \gamma$

Example. For $\gamma = 2$, an edge $(\ell, \ell', g, \sigma, R)$ in A with $g \equiv x = 4 \wedge 1 \leq y \leq 3$ is replaced in A_γ by the four edges:

$$(\ell, \ell', \{x = 8, 2 \leq y \leq 3\}, \sigma, R) \qquad (\ell, \ell', \{x = 8, 4 \leq y \leq 5\}, \sigma, R)$$
$$(\ell, \ell', \{x = 8, 3 \leq y \leq 4\}, \sigma, R) \qquad (\ell, \ell', \{x = 8, 5 \leq y \leq 6\}, \sigma, R)$$

Theorem 6.40 *Let A be a closed timed automaton with $n \geq 1$ clocks and largest constant M. For all distances $0 < \alpha < 1$, for all number of steps $k \in \mathbb{N}$, there exist two numbers $D, E \in \mathbb{R}^{>0}$ such that for all $\Delta \in [0, D]$, for all $\varepsilon \in [0, E]$ and for all trajectories π' of $[\![A]\!]^\varepsilon_\Delta$ in normal form such that $|\pi'| = k$, there exists a trajectory π of $[\![A]\!]$ such that:*

- $\mathsf{first}(\pi) \in [\mathsf{first}(\pi')]$;
- $\mathsf{trace}(\pi) = \mathsf{trace}(\pi')$;
- *π is "close" to π': $\forall\, 0 \leq i \leq k$: if $\mathsf{state}_i(\pi) = (\ell_i, v_i)$ and $\mathsf{state}_i(\pi') = (\ell'_i, v'_i)$ then $\ell_i = \ell'_i$ and $\|v_i - v'_i\|_\infty < \alpha$.*

Proof. Given $0 < \alpha < 1$ and $k \in \mathbb{N}$, let $\gamma = \lceil 2/\alpha \rceil$ and:

$$D = \frac{\alpha}{4\gamma(n+1)^{k+1}} \qquad\qquad E = \frac{D}{2(\gamma M + 1)}$$

Let $\Delta \in [0, D]$ and $\varepsilon \in [0, E]$ and let $\Omega = \gamma D$. Let $\mathsf{trace}(\pi') = \sigma_1\sigma_2\ldots\sigma_k$ be the trace of π'. Let $\rho' = \mu_\gamma(\pi')$. Then ρ' is a trajectory of $[\![A_\gamma]\!]^{\varepsilon}_{\gamma\Delta}$. Let $\mathbf{M}_0$ be a PDBM in normal form such that $[\![\mathbf{M}_0]\!] = [\mathsf{first}(\pi')]$ and $w(\mathbf{M}_0) = 0$ (in fact $\mathbf{M}_0$ is a DBM). Observe that $\Delta \leq \Omega$ and $\varepsilon \leq \Omega/(2(M_\gamma + 1))$ where $M_\gamma = \gamma M$ is the largest constant of A_γ. Therefore, by Lemma 6.38 and 6.39, there exists PDBM $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_k$ in normal form such that for all $1 \leq i \leq k$:

- (a) $\mathsf{post}^{\sigma_i}_{[\![A_\gamma]\!]^{\varepsilon}_{\gamma\Delta}}([\![\mathbf{M}_{i-1}]\!]_\Omega) \subseteq [\![\mathbf{M}_i]\!]_\Omega$;
- (b) $\mathsf{post}^{\sigma_i}_{[\![A_\gamma]\!]}([\![\mathbf{M}_{i-1}]\!]_0) = [\![\mathbf{M}_i]\!]_0$;
- (c) $w(\mathbf{M}_i) \leq \max\{w(\mathbf{M}_{i-1}) + 1, n \cdot \max\{2, w(\mathbf{M}_{i-1})\}$.

Let us show that $w(\mathbf{M}_i) \leq 2(n+1)^i$. We proceed by induction. The claim holds for $i = 1$ since $w(\mathbf{M}_0) = 0$. Assume that it holds for $i-1$ for $i \geq 2$. Then we have:

$$\begin{aligned} w(\mathbf{M}_i) &\leq \max\left\{w(\mathbf{M}_{i-1}) + 1,\ n \cdot \max\{2, w(\mathbf{M}_{i-1})\}\right\} \\ &\leq \max\{1 + 2(n+1)^{i-1},\ 2n \cdot (n+1)^{i-1}\} \quad \text{by } (c) \\ &\leq 2(n+1)^{i-1} + 2n \cdot (n+1)^{i-1} \\ &\leq 2(n+1)^i \end{aligned}$$

For each $0 \leq i \leq k$, let $q'_i = \mathsf{state}_i(\pi')$. By (a), we have $\mu_\gamma(q'_k) \in [\![\mathbf{M}_k]\!]_\Omega$. Since $\alpha < 1$, it is easy to see that:

$$\Omega = \frac{\alpha}{4(n+1)^{k+1}} < \frac{1}{(2n+1)w(\mathbf{M}_k)}$$

and thus by Lemma 6.36, there exists $q_k \in [\![\mathbf{M}_k]\!]$ such that:

$$\|\mu_\gamma(q'_k) - q_k\|_\infty \leq n \cdot w(\mathbf{M}_k) \cdot \Omega < 2(n+1)^{k+1} \cdot \Omega \leq \alpha$$

Using (b), we can construct in a backward fashion a trajectory ρ of $[\![A_\gamma]\!]$ such that:

- $\mathsf{last}(\rho) = q_k$;
- $\mathsf{trace}(\rho) = \mathsf{trace}(\pi')$;
- $\mathsf{first}(\rho) \in [\![\mathbf{M}_0]\!] = [q'_0]$.

For each $0 \leq i \leq k$, let $q_i = \mathsf{state}_i(\rho)$. For all i such that $\sigma_i \neq \mathsf{time}$, we have $q_i \in [\![g'_i]\!]$ and $\mu_\gamma(q'_i) \in \mathcal{N}_\infty([\![g'_i]\!], \gamma\Delta)$ where g'_i is the guard of the edge of A_γ associated to σ_i that has been taken in ρ. Since the size of g'_i is at most 1, we have:

$$\|\mu_\gamma(q'_i) - q_i\|_\infty \leq 1 + \gamma\Delta \leq 1 + \Omega \tag{6.9}$$

Observe that the effect of discrete transitions is to reset some clocks and that does not increase the ∞-distance between two states: we also have $\|\mu_\gamma(q'_i) - q_i\|_\infty \leq 1 + \gamma\Delta$ for all i such that $\sigma_{i-1} \neq \mathsf{time}$. Since π' is in normal form and $\mathsf{trace}(\rho) = \mathsf{trace}(\pi')$, Equation (6.9) holds for all $0 \leq i \leq k$. Now, let $\pi = \mu_\gamma^{-1}(\rho)$ which is a trajectory of $[\![A]\!]$ since ρ is a trajectory of $[\![A_\gamma]\!]$. Thus, we have for all $0 \leq i \leq k$:

$$\|q'_i - \mu_\gamma^{-1}(q_i)\|_\infty \leq \frac{1+\Omega}{\gamma} < \frac{2}{\gamma} \leq \alpha$$

which entails that π is "close" to π' as required. ■

The following theorem is the key of the proof of completeness. It shows that for all distances $\alpha > 0$, we can choose sufficiently small values of Δ and ε such that from J^* the points that are reachable in $[\![A]\!]^\varepsilon_\Delta$ are at distance at most α from J^*. By contrast with Theorem 6.40, we do not make the hypothesis that the length of the trajectories is bounded. This result is similar to Theorem 8.3 in [Pur98], but the constants are different because only drifting clocks were considered by Puri and the bound of Lemma 6.12 was wrong.

Theorem 6.41 *Let A be a closed timed automaton with $n \geq 1$ clocks and largest constant M that satisfies Assumption 6.4. For all distances $\alpha \in \mathbb{R}^{>0}$, there exist two numbers $D, E \in \mathbb{R}^{>0}$ such that for all $\Delta \in [0, D]$, for all $\varepsilon \in [0, E]$ and for all trajectories π' of $[\![A]\!]^\varepsilon_\Delta$ such that $\mathsf{first}(\pi') \in J^*$, we have $d_\infty(\mathsf{last}(\pi'), J^*) < \alpha$.*

Proof. Without loss of generality, we may assume that $\alpha < \frac{1}{2n}$. Let W be the number of regions of A, let $\gamma = \lceil 2/\alpha \rceil$ and:

$$D = \frac{\alpha}{4\gamma(n+1)^{2W+1}} \qquad\qquad E = \frac{D}{2(\gamma M + 1)}$$

Let $\Delta \in [0, D]$ and $\varepsilon \in [0, E]$ and let π' be a trajectory of $[\![A]\!]^\varepsilon_\Delta$ in normal form such that $\mathsf{first}(\pi') \in J^*$. Let $m = |\pi'|$ and for each $0 \leq i \leq m$, let $q'_i = state_i(\pi')$.

- If $m \leq 2W$. By Theorem 6.40, there exists a trajectory π of $[\![A]\!]$ such that $\mathsf{first}(\pi) \in [\mathsf{first}(\pi')]$ and for all $0 \leq i \leq m$, $\|q_i - q'_i\|_\infty < \alpha$ where $q_i = state_i(\pi)$. Since $q_0 \in [q'_0] \subseteq J^*$, the state q_m is reachable from J^* and thus $q_m \in J^*$. Since $\|q_m - q'_m\|_\infty < \alpha$ this yields $d_\infty(\mathsf{last}(\pi'), J^*) < \alpha$.

- If $m > 2W$. By induction, assume that $d_\infty(q'_i, J^*) < \alpha$ for all $0 \leq i \leq m-1$. Consider the sub-trajectory of π' from state q'_{m-2W} to q'_m, and according to Theorem 6.40 let π be a trajectory such that for all $m-2W \leq i \leq m$, $\|q_i - q'_i\|_\infty < \alpha$ where $q_i = state_{i-(m-2W)}(\pi)$. Then for all i, $m - 2W \leq i \leq m-1$, we have :

$$d_\infty(q_i, J^*) \leq \|q_i - q'_i\|_\infty + d_\infty(q'_i, J^*) \leq 2\alpha < \frac{1}{n}$$

and by Lemma 6.12, this implies $[q_i] \cap J^* \neq \varnothing$ for all $m-2W \leq i \leq m-1$ (since J^* is a zone-set).

On the other hand, the trajectory π has the same trace as the sub-trajectory of π' from q'_{m-2W} to q'_m and thus it is in normal form and $|\pi| = 2W$. Therefore, π has $2W+1$ states and thus there exists two states q_k and $q_{k'}$ in π with $k < k'$ such that $[q_k] = [q_{k'}]$ and a discrete occurred along π between q_k and $q_{k'}$ in π, and thus there exists a path from $[q_k]$ to itself in the region automaton of A. Since $[q_k] \cap J^* \neq \varnothing$, we have $[q_k] \subseteq J^*$ by line 6 of Algorithm 4 and $[q_i] \subseteq J^*$ for all $i \geq k$ by line 7 of the algorithm. So we have $q_m \in J^*$ and since $\|q_m - q'_m\|_\infty < \alpha$, this yields $d_\infty(\mathsf{last}(\pi'), J^*) < \alpha$.

■

6.6.3.1 Completeness of Algorithm 4.

Theorem 6.42 *Let J^* be the set computed by Algorithm 4. Under Assumption 6.4, we have $R^*_{\Delta,\varepsilon} \subseteq J^*$.*

Proof. For all $y \in R^*_{\Delta,\varepsilon}$, for all $\Delta > 0$ and $\varepsilon > 0$ there exists a trajectory π of $[\![A]\!]^\varepsilon_\Delta$ such that $\mathsf{first}(\pi) \in J^*$ (because J^* contains the initial states) and $\mathsf{last}(\pi) = y$. Therefore, by Theorem 6.41 for all $\alpha \in \mathbb{R}^{>0}$ we have $d_\infty(y, J^*) < \alpha$. This implies that $d_\infty(y, J^*) = 0$ and since J^* is a closed set (a finite union of closed regions) we have $y \in J^*$. ■

With Theorem 6.35 and Theorem 6.42 we have proven the following inclusions.

$$R^*_{\Delta,\varepsilon} \subseteq \begin{array}{c} \subsetneq \\ J^* \\ \subseteq \end{array} \begin{array}{c} R^*_\varepsilon \\ \\ R^*_\Delta \end{array} \begin{array}{c} \subseteq \\ \\ \subsetneq \end{array} R^*_{\Delta,\varepsilon}$$

All those sets are thus equal:

Theorem 6.43 *Under Assumption 6.4, we have $R^*_\Delta = R^*_\varepsilon = R^*_{\Delta,\varepsilon}$, and those sets are computed by Algorithm 4.*

6.6.4 Complexity

The complexity issues have been studied in [Pur98]. We mention the main theorem and we give a detailed proof of the hardness result.

Theorem 6.44 ([Pur98]) *Given a timed automaton* $A = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ *satisfying Assumption 6.4 and a location* $\ell \in \mathsf{Loc}$*, deciding whether there exists a valuation* v *such that* $(\ell, v) \in R^*_\Delta$ *(or equivalently* $(\ell, v) \in R^*_\varepsilon$*, or* $(\ell, v) \in R^*_{\Delta,\varepsilon}$*) is* PSPACE-COMPLETE*.*

The proof uses the following definition of Linear Bounded Turing Machines (LBTM). A LBTM is a nondeterministic Turing machine that can only use a number of tape cells equal to the length of its input.

Definition 6.45 [Linear Bounded Turing Machine] A *LBTM* $M = (Q, \Sigma, q_0, q_f, E)$ consists of:

- a finite set of control states Q,
- a finite alphabet Σ,
- an initial state $q_0 \in Q$, a final state $q_f \in Q$,
- and a set of transitions $E \subseteq Q \times \Sigma \times \Sigma \times \{\mathsf{left}, \mathsf{right}\} \times Q$.

A *configuration* of M is a triple $(q, w, i) \in Q \times \Sigma^* \times \mathbb{N}$ where q is a control location, $w \in \Sigma^*$ is the content of the tape, and i is the position of the tape head. A configuration (q', w', i') is a *successor* of a configuration (q, w, i) iff there exists a transition $(q, \sigma, \sigma', d, q') \in E$ such that:

(1) $w_i = \sigma$;

(2) $w'_i = \sigma'$ and $w'_j = w_j$ for all $j \neq i$;

(3) $i' = i - 1$ if $d = \mathsf{left}$ and $i' = i + 1$ if $d = \mathsf{right}$ with $1 \leq i' \leq |w|$.

We assume that the condition $1 \leq i' \leq |w|$ is realized using input delimiters. An *execution* of M on the input $x \in \Sigma^*$ is a sequence $s_0 s_1 \dots s_n$ of configurations starting with $s_0 = (q_0, x, 1)$ and such that s_{i+1} is a successor of s_i for every $0 \leq i < n$. We say that M *accepts* x iff M has an execution on x finishing in $s_n = (q_f, w, i)$ for some $w \in \Sigma^*$ and $i \in \mathbb{N}$. The *acceptance problem for LBTM* asks, given a LBTM M and an input word $x \in \Sigma^*$ whether M accepts x. □

Proof of Theorem 6.44. First, we prove PSPACE-membership. It is not possible to use Algorithm 4 because we should construct the region automaton R_A, which may have a number of states exponential in the number of clocks of the timed automaton A. However, we can check the reachability of a region r by guessing a path in the region automaton from the initial regions to r in polynomial space. This is a fairly standard trick used for showing PSPACE-membership of the reachability problem for

classical timed automata with an on-the-fly algorithm [AD94]. The difficulty is that the successor of a given region r can be a neighbour region r' such that $[r] \cap [r'] \neq \varnothing$ provided r' lies in a progress cycle S of R_A. As we have shown, the entire region r' can be reached from r in $[\![A]\!]_\Delta$ no matter how small is Δ, by repeating the cycle S. Hence we can add S in one step in the set of reachable states. Such an *acceleration* has been proven correct (Theorem 6.35) and complete (Theorem 6.42). So, when guessing the successor of a region r, we must take into account the neighbour regions of r and decide whether they are in a progress cycle or not. This can be checked in PSPACE using the same procedure as for classical timed automata [AD94]. A polynomially bounded part of the memory is reserved for executions of this procedure. Since the content of this part of the memory is not necessary for further computations, it can be reused by subsequent calls and PSPACE-membership follows.

We establish PSPACE-hardness using a reduction of the acceptance problem for LBTM which is known to be PSPACE-hard [Kar72].

Our reduction is similar to [CY91], where a configuration (q, w, i) of a LBTM is encoded by a location (q, i) (that records the control state q and the tape position i) and by the clocks $y_1, \ldots, y_{|w|}$, one for each tape cell. We assume without loss of generality that $\Sigma = \{a, b\}$. A clock y_i has the value $y_i = n_a$ if $w_i = a$ and $y_i = n_b > n_a$ if $w_i = b$. This encoding is not preserved by time passing. Thus we need to periodically refresh the values of the clocks. This is done in two phases: (I) resetting the clocks coding a 'b' (by checking $y_i = n_b$), then letting $n_b - n_a$ time unit pass, and (II) resetting the clock coding an 'a' (by checking $y_i = n_b$ again) and finally letting n_a time unit pass. During phase (I), the clock that encodes the tape cell pointed by the head, is updated according to the transitions of the LBTM.

We show how to adapt this reduction to the enlarged semantics of timed automata. Due to guards enlargements, equality cannot be tested precisely and the clocks can not store precise values n_a and n_b. However, if Δ is sufficiently small and n_a and n_b are not too close, we can still distinguish clocks coding an 'a' and clocks coding a 'b'. The main details of the proof follow.

Let $M = (Q, \Sigma, q_0, q_f, \delta)$ be a LBTM and $x \in \Sigma^*$ be an input word. Let $n = |x|$, $n_a = 3$ and $n_b = 6$. We construct a timed automaton $\mathcal{A}(M, x)$ with $n + 1$ clocks and a location ℓ_f such that M accepts x iff $(\ell_f, v) \in R^*_\Delta$ for some valuation v. Let $\mathcal{A}(M, x) = \langle \mathsf{Loc}, \mathsf{Var}, \mathsf{Init}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg}, \mathsf{Final} \rangle$ with:

- $\mathsf{Loc} = \{s_0, s_1, \ell_f\} \cup \{(q, i, j, \phi, d) \mid q \in Q \wedge 1 \leq i \leq n \wedge 1 \leq j \leq n + 1 \wedge \phi \in \{\mathrm{I}, \mathrm{II}\} \wedge d \in \{\mathsf{left}, \mathsf{right}\}\}$; a location (q, i, j, ϕ, d) encodes the control state q, the tape position i, the number j of the next clock to be treated, the phase ϕ of the simulation, and the direction d of the next head movement;
- $\mathsf{Var} = \{y_i \mid 1 \leq i \leq n\} \cup \{z\}$;
- $\mathsf{Init}(s_0) \equiv \bigwedge_{t \in \mathsf{Var}} t = 0$ and for all $\ell \in \mathsf{Loc} \setminus \{s_0\} : \mathsf{Init}(\ell) = \bot$;

- $\mathsf{Inv}(\ell) = \top$ and $\mathsf{Final}(\ell) = \bot$ for all $\ell \in \mathsf{Loc}$;
- $\mathsf{Lab} = \{\tau\}$;
- The set Edg contains the following edges (we write $\ell \xrightarrow{g,R} \ell'$ when $(\ell, \ell', g, \tau, R) \in \mathsf{Edg}$):

– *Initialization*:

  - $s_0 \xrightarrow{z=3,\{y_i \mid x_i = a\} \cup \{z\}} s_1$
  - $s_1 \xrightarrow{z=3,\{z\}} (q_0, 1, 1, I, \mathsf{left})$

– *Refresh*: for every $(q, i, j, \phi, d) \in \mathsf{Loc}$ with $j \neq i$ and $j \leq n$,

  - $(q, i, j, \phi, d) \xrightarrow{z \leq 0 \wedge y_j \leq 4, \varnothing} (q, i, j+1, \phi, d)$
  - $(q, i, j, \phi, d) \xrightarrow{z \leq 0 \wedge y_j \geq 5, \{y_j\}} (q, i, j+1, \phi, d)$
  - $(q, i, i, \mathrm{II}, d) \xrightarrow{z \leq 0 \wedge y_i \leq 4, \varnothing} (q, i, i+1, \mathrm{II}, d)$
  - $(q, i, i, \mathrm{II}, d) \xrightarrow{z \leq 0 \wedge y_i \geq 5, \{y_i\}} (q, i, i+1, \mathrm{II}, d)$

– *Execution*:for every $q \in Q$, $1 \leq i \leq n$, $d \in \{\mathsf{left}, \mathsf{right}\}$, and for every transition $(q, \sigma, \sigma', d', q') \in E$,

  - If $(\sigma, \sigma') = (a, a)$ then $(q, i, i, \mathrm{I}, d) \xrightarrow{z \leq 0 \wedge y_i \leq 4, \varnothing} (q', i, i+1, \mathrm{I}, d')$
  - If $(\sigma, \sigma') = (a, b)$ then $(q, i, i, \mathrm{I}, d) \xrightarrow{z \leq 0 \wedge y_i \leq 4, \{y_i\}} (q', i, i+1, \mathrm{I}, d')$
  - If $(\sigma, \sigma') = (b, a)$ then $(q, i, i, \mathrm{I}, d) \xrightarrow{z \leq 0 \wedge y_i \geq 5, \varnothing} (q', i, i+1, \mathrm{I}, d')$
  - If $(\sigma, \sigma') = (b, b)$ then $(q, i, i, \mathrm{I}, d) \xrightarrow{z \leq 0 \wedge y_i \geq 5, \{y_i\}} (q', i, i+1, \mathrm{I}, d')$

– *Phase change*: for every $q \in Q$, $1 \leq i \leq n$, $j = n+1$ and $d \in \{\mathsf{left}, \mathsf{right}\}$,

  - $(q, i, n+1, \mathrm{I}, d) \xrightarrow{z=3,\{z\}} (q, i, 1, \mathrm{II}, d)$
  - $(q, i, n+1, \mathrm{II}, \mathsf{left}) \xrightarrow{z=3,\{z\}} (q, i-1, 1, \mathrm{I}, \mathsf{left})$
  - $(q, i, n+1, \mathrm{II}, \mathsf{right}) \xrightarrow{z=3,\{z\}} (q, i+1, 1, \mathrm{I}, \mathsf{right})$

– *Termination*: for every $1 \leq i \leq n$, $d \in \{\mathsf{left}, \mathsf{right}\}$,

  - $(q_f, i, 1, \mathrm{I}, d) \xrightarrow{\top, \varnothing} \ell_f$

After the *initialization step*, the automaton is in the location $(q_0, 1, 1, I, \mathsf{left})$ and we have the following relation between the tape content w and the clocks $y_1, \ldots, y_n$ when $z = 0$:

$$\begin{cases} 3 - \Delta \leq y_i \leq 3 + \Delta & \text{if } w_i = a \\ 6 - 2\Delta \leq y_i \leq 6 + 2\Delta & \text{if } w_i = b \end{cases}$$

After executing one transition $(q, \sigma, \sigma', d', q')$ of M, let w' be the new tape content (w' differs from w by at most one symbol). If we simulate that transition by the *refresh* steps, the *execution* step, and the *phase changes*, it is easy to check that in location $(q, i, 1, I, d)$, when $z = 0$ we have:

$$\begin{cases} 3 - 2\Delta \leq y_i \leq 3 + \Delta & \text{if } w'_i = a \\ 6 - 3\Delta \leq y_i \leq 6 + 2\Delta & \text{if } w'_i = b \end{cases} \tag{6.10}$$

Note that two clocks coding the same symbol are not necessarily equal (however, their difference is bounded by Δ). The reader can check that after having executed a second transition of M, there is no accumulation of the imprecisions and the conditions (6.10) still hold. Hence, provided Δ is sufficiently small (in fact $\Delta < 1/2$), the automaton $\mathcal{A}(M, x)$ will correctly distinguish clocks coding 'a' from clocks coding 'b' for any number of transitions, and thus simulate faithfully the execution of M on x. It is now easy to see that the location ℓ_f is reachable in R^*_Δ iff ℓ_f is reachable in $[\![\mathcal{A}]\!]^0_0$ iff M accepts x. This concludes the proof since our construction is polynomial in the size of M and x. ■

6.7 Beyond Progress Cycles

We review some classical extensions of the model of closed timed automata, and their implication on the decidability result of the implementability problem. We have already mentioned (after Definition 6.2) that the result still holds for Alur-Dill automata, where both loose and strict inequalities are allowed. We can go further and claim that the result holds for timed automata over rectangular and diagonal constraints (*i.e.* constraints of the form $x - y \leq a$ or $x - y < a$ where $a \in \mathbb{N}$). The definition of the enlarged semantics need not to be modified as we use the notion of neighbourhood to define guard enlargements. Remind of the fact that all the points in a region of a timed automaton satisfy the same set of rectangular and diagonal constraints. In several contexts, to show that a valuation v satisfies a guard g, we have often used the fact that $v \in r \wedge r \models g$ for some region r. This argument is obviously valid for diagonal constraints. The other proofs are easily adapted: for Lemma 6.25, we have used an argument of the form $v \models g \wedge v' \in \mathcal{N}_\infty(v, \eta)$ to show that $v' \in \mathcal{N}_\infty([\![g]\!], \eta)$ which is fairly independent of the form of g. Finally, the proof of completeness was based on properties of PDBM and the fact that guards can be encoded by PDBM of width 2

(page 132). It is easy to see that diagonal constraints are representable by PDBM, with the same property.

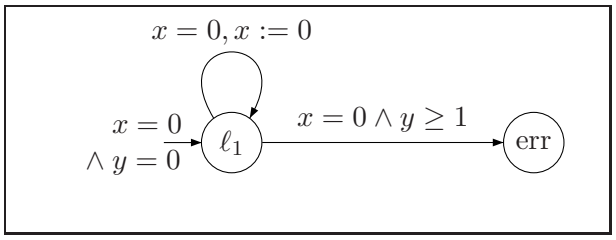


Figure 6.7: A closed timed automaton that does not satisfy Assumption 6.4.

Now, consider Assumption 6.4 about progress cycles. Figure 6.7 shows a closed timed automaton whose region automaton has a non progress cycle (the self-loop on location ℓ_1 does not reset y). Algorithm 4 would compute $J^* = \{(\ell_1, v) \mid v(x) = v(y)\}$. However, when $\Delta > 0$ no matter how small, the loop on ℓ_1 can increase y unboundedly, and thus the location err is reachable in $[\![A]\!]_\Delta$ for all $\Delta > 0$. Therefore, $J^* \neq R^*_\Delta$. On the other hand, if the only perturbation of the classical semantics is drifts on clocks, then Algorithm 4 is correct on this example: $J^* = R^*_\varepsilon$. We conjecture that it is always true that Assumption 6.4 is not necessary for drifting clocks. For imprecise guards, we conjecture that the following adaptation of the algorithm solves the problem: for non progress cycles, instead of adding the whole region $[p_0]$ that intersects J^* (as in line 6 of Algorithm 4), we could add only the part of $[p_0]$ that is reachable from $J^* \cap [p_0]$ by time passing with first derivative equal to zero for the clocks that are reset in the cycle. Thus, we would replace line 6 by the assignment $J^* \leftarrow J^* \cup ([p_0] \cap ([p_0] \cap J^*)^{\nearrow}_{X_S})$ where X_S is the set of clocks that are reset at least once in the cycle S, and $r^{\nearrow}_X = \{v' \mid \exists t \in \mathbb{R}^{\geq 0} \cdot \exists v \in r : v'(x) = v(x)$ if $x \in X$ and $v'(x) = v(x) + t$ otherwise $\}$. This set is a union of regions. We have no proof of those conjectures, but it may appear useless as the kind of timed automata that are excluded by Assumption 6.4 have no interest in practice (see for instance Figure 6.7, where a clock is both tested and reset to 0).

6.8 Conclusion

In this chapter, we have studied an algorithm proposed by Puri to compute a semantics of timed automata that is robust against drift on clocks, and we have shown that it also computes a robust semantics against perturbations of the guards. We have also drawn an important link between robustness and implementability of timed automata to show that implementability problem is decidable.

The key feature of the algorithm is to compute the cycles of the region automaton. It can be seen as defining exact accelerations of the cycles of the timed automaton. Since constructing the region automaton is unrealistic in practice, the algorithm is not

well suited for practical use. A necessary future work is to design a more practical algorithm, for example with a symbolic computation of the cycles, similar to what exists for reachability analysis with DBM or BDD.

As a future work, it may also be interesting to prove the conjecture that the maximal value of Δ such that $[\![A]\!]^0_\Delta$ is empty, is a rational number.

Chapter 7

Verification of Affine Hybrid Automata

Sans la liberté de blâmer, il n'est point d'éloge flatteur.

Beaumarchais, *La folle journée ou le mariage de Figaro*.

7.1 Introduction

In the previous chapters, we have studied the robustness and implementability properties of timed automata. This was motivated by the fact that in real-time systems where an environment is controlled by a digital device, a natural model for the digital controller is a timed automaton: the finite structure encodes the discrete states of the controller, and the clocks constrain the timing behaviour. Simple dynamics of the form $\dot{x} = 1$ are sufficient because real controllers refer to time through their internal *clock*. For the second part of the system, the environment, the context is different. In general, the environment models a physical system whose behaviour is governed by non trivial differential equations. As we have mentioned in Section 2.5, the emptiness problem is undecidable for more general than uniformly constant dynamics, like in timed automata.

Therefore, the usual way to analyze hybrid automata is by way of approximations. A general methodology has been proposed in [HHW98] where rectangular automata are shown to have the property that they can over-approximate, at any level of precision, the set of behaviors of more complex hybrid automata. For rectangular automata, there exists a reasonably efficient semi-algorithm to compute the set of reachable states [ACH+95], and when the semi-algorithm terminates, the correctness of the system is easily decided. The methodology can be summarized as follows: to establish a safety property on a complex hybrid automaton A, construct a rectangular approximation B of A and check the safety property on B. If the property is estab-

lished on B, then it also holds on A. If the property is not established on B, then use a more precise rectangular approximation B'.

So far, the construction of the abstraction B and its refinement B' was supposed to be obtained manually. In this chapter, we show how to efficiently automate this methodology for the class of affine hybrid automata, that is hybrid automata whose continuous dynamics are defined by systems of linear differential equations. More precisely, we show (i) how to compute automatically rectangular approximations for affine hybrid automata, (ii) how to refine automatically and in an optimal way rectangular approximations that fail to establish a given safety property, and (iii) how to target refinement only to relevant parts of the state space.

Refinements are obtained by splitting location invariants. A split is optimal if it minimizes some measure of the imprecision of the resulting rectangular approximation. Intuitively, the imprecision corresponds to the size of the interval defining the rectangular flow, because the smaller the imprecision, the closer the rectangular dynamics is to the exact dynamics. We choose to minimize the maximal imprecision of the dynamics in the split location of the refined automaton.

7.2 Preliminaries

The following definitions are based on the preliminaries of Section 2.1 and on the hybrid automata of Section 2.5. In this chapter, the time domain $\mathbb{T}$ is always the set of nonnegative real numbers $\mathbb{R}^{\geq 0}$.

Given a function $f : A \to B$ and $b \in B$, define the *level set* of f corresponding to b by $f^{-1}(b) = \{a \in A \mid f(a) = b\}$. For $C \subseteq B$, define $f^{-1}(C) = \{a \in A \mid f(a) \in C\}$. For a function $f\colon \mathbb{R}^n \to \mathbb{R}$, we denote by ∇f the vector $(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$.

Predicates Given two rectangular predicates $p = \bigwedge_{x \in X} x \in I_x$ and $q = \bigwedge_{x \in X} x \in J_x$, we define the *size* (or diameter) of p by $|p| = \max_{x \in X}\{\mathsf{size}(I_x)\}$. It is the length of the largest interval defining p. The *rectangular hull* of p and q is the predicate $p \sqcup q = \bigwedge_{x \in X} x \in I_x \sqcup J_x$.

Lines and hyperplanes A function $f : \mathbb{R}^n \to \mathbb{R}$ is called *affine* if it is of the form $f(x) = a_0 + \sum_i a_i x_i$ with $a_i \in \mathbb{Q}$, for all $0 \leq i \leq n$. We say that two affine functions f_1 and f_2 are *parallel* if for some $\lambda \in \mathbb{R}$ the function $f(x) = f_1(x) + \lambda f_2(x)$ is independent of x (that is, ∇f is identically 0). A *hyperplane* is a level set of an affine function. Given an affine function $f : \mathbb{R}^n \to \mathbb{R}$, we write $\pi \equiv f(x) = 0$ to denote the hyperplane $\pi = f^{-1}(0)$. In $\mathbb{R}^2$ a hyperplane is a *line*. Given two points $a, b \in \mathbb{R}^2$, let $\ell ine(a, b)$ denote the line passing by a and b. We write $[a, b]$ for the *line segment* connecting a and b, *i.e.* the set of convex combinations $\{\lambda a + (1 - \lambda)b \mid 0 \leq \lambda \leq 1\}$.

Simulations We have defined simulation and weak simulation for TTS in Section 2.3. In the sequel, we also use a slightly different notion of simulation where only the unsafe states are to be simulated, because of the following observation: if a state q has been proven to be safe in $\mathcal{T}_1$ (that is $q \notin \mathsf{Unsafe}(\mathcal{T}_1)$) and $\mathcal{T}_1 \succeq \mathcal{T}_2$ with the simulation relation R, then all the states q' such that $(q, q') \in R$ are necessarily safe in $\mathcal{T}_2$. Therefore the emptiness problem for an hybrid automaton H has the same answer as the emptiness problem on H with some of its safe states removed. So, it is sufficient to refine the automaton in its unsafe states. More details are given in Section 7.5.

Definition 7.1 [Weak simulation for unsafe behaviours] Given two TTS $\mathcal{T}_1 = \langle Q^1, Q_0^1, Q_f^1, \Sigma^1, \rightarrow^1 \rangle$ and $\mathcal{T}_2 = \langle Q^2, Q_0^2, Q_f^2, \Sigma^2, \rightarrow^2 \rangle$, we write $\mathcal{T}_1 \succeq_{unsafe} \mathcal{T}_2$ and say that $\mathcal{T}_1$ *weakly simulates the unsafe behaviours of* $\mathcal{T}_2$ if there exists a relation $R \subseteq Q^1 \times \mathsf{Unsafe}(\mathcal{T}_2)$ such that:

1. for all $(q_1, q_2) \in R$, for each $\sigma \in \Sigma \backslash \{\tau\} \cup \mathbb{R}^{\geq 0}$, if $q_2 \stackrel{\sigma}{\twoheadrightarrow} q_2'$ and $q_2' \in \mathsf{Unsafe}(\mathcal{T}_2)$, then there exists $q_1' \in Q^1$ such that $q_1 \stackrel{\sigma}{\twoheadrightarrow} q_1'$ and $(q_1', q_2') \in R$,
2. for all $q_2 \in Q_0^2 \cap \mathsf{Unsafe}(\mathcal{T}_2)$, there exists $q_1 \in Q_0^1$ such that $(q_1, q_2) \in R$,
3. and for all $q_2 \in Q_f^2 \cap \mathsf{Unsafe}(\mathcal{T}_2)$, for all $q_1 \in Q^1$, if $(q_1, q_2) \in R$, then $q_1 \in Q_f^1$.

Such a relation R is called a *weak simulation relation* for $\mathcal{T}_1 \succeq_{unsafe} \mathcal{T}_2$. □

Definition 7.2 [Weak bisimulation for unsafe behaviours] Two TTS $\mathcal{T}_1$ and $\mathcal{T}_2$ are *weakly bisimilar for unsafe behaviours* (noted $\mathcal{T}_1 \approx_{unsafe} \mathcal{T}_2$) if there exists a simulation relation R for $\mathcal{T}_1 \succeq_{unsafe} \mathcal{T}_2$ and a simulation relation S for $\mathcal{T}_2 \succeq_{unsafe} \mathcal{T}_1$ such that $R = S^{-1}$. □

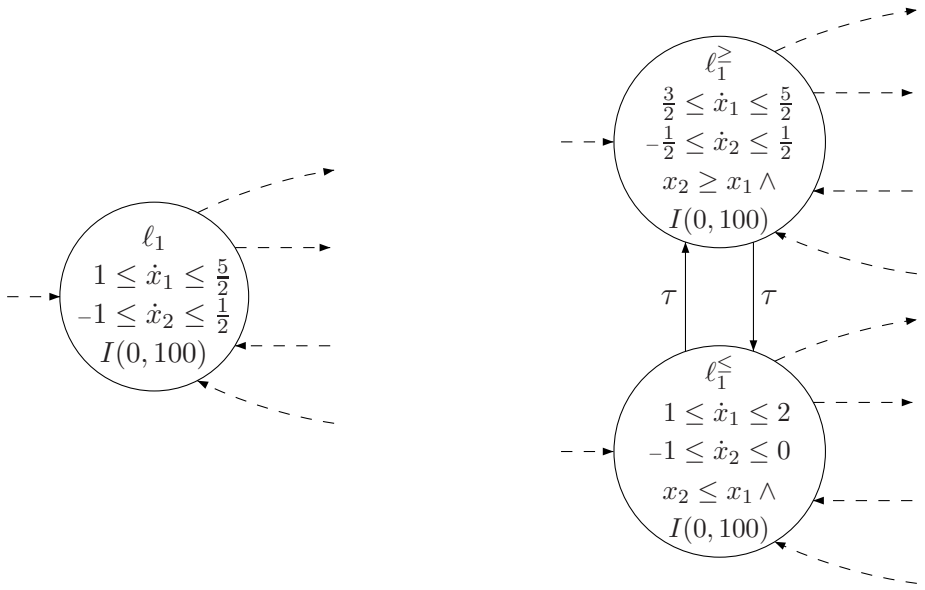
Lemma 7.3 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two* TTS*. If $\mathcal{T}_1 \succeq \mathcal{T}_2$, then $\mathcal{T}_1 \succeq_{unsafe} \mathcal{T}_2$.*

Theorem 7.4 *Let H and H' be two hybrid automata such that $[\![H']\!] \succeq_{unsafe} [\![H]\!]$. If $[\![H']\!]$ is empty, then so is $[\![H]\!]$.*

Over-approximations

Definition 7.5 [Rectangular phase-portrait approximation] We say that a hybrid automaton H' is a *rectangular phase-portrait approximation* of an hybrid automaton H if $[\![H']\!] \succeq [\![H]\!]$ and H' is a rectangular automaton. □

A natural way of constructing a rectangular phase-portrait approximation $H' = \mathsf{rect}(H)$ of an affine automaton H is to replace in each location ℓ of H the affine flow condition $\mathsf{Flow}_H(\ell) = \bigwedge_{x \in X} \dot{x} = t_x$ by the rectangular predicate $\mathsf{Flow}_{H'}(\ell) = \bigwedge_{x \in X} \dot{x} \in I_x$ where I_x is the convex hull of the set $\{[\![t_x]\!]_v \mid v \in [\![\mathsf{Inv}(l)]\!]\}$. The bounds of I_x can be determined by a linear program since t_x is a linear term and $\mathsf{Inv}(l)$ is a linear predicate. The proof that $[\![H']\!] \succeq [\![H]\!]$ is immediate since for any $v \in [\![\mathsf{Flow}_H(\ell)]\!]$ we have $v_{|\dot{X}} \in [\![\mathsf{Flow}_{H'}(\ell)]\!]$.



(a) Rectangular phase-portrait.

(b) Location splitting.

Figure 7.1: Location ℓ_1 of the shared gas-burner.

Example Figure 7.1(a) depicts the result of this construction when applied to the location ℓ_1 of the shared gas-burner automaton of Figure 2.1. For x_1, since $\dot{x}_1 = h_1 - a_1 x_1 + b_1 x_2$ and the invariant of ℓ_1 imposes $x_1, x_2 \in [0, 100]$, a lower bound for $\dot{x}_1$ is $h_1 - 100 \cdot a_1 = 1$ and an upper bound is $h_1 + 100 \cdot b_1 = \frac{5}{2}$. The process is similar for $\dot{x}_2$.

For an hybrid automaton H with set of locations Loc_H, let $\mathsf{SafeLoc}(H) = \{\ell \in \mathsf{Loc}_H \mid \nexists (\ell, v) \in \mathsf{Unsafe}([\![H]\!])\}$.

Lemma 7.6 *For every affine automaton H, we have* $\mathsf{SafeLoc}(\mathsf{rect}(H)) \subseteq \mathsf{SafeLoc}(H)$.

Proof. For any location $\ell \notin \mathsf{SafeLoc}(H)$, let us show that $\ell \notin \mathsf{SafeLoc}(\mathsf{rect}(H))$. By definition of $\mathsf{SafeLoc}(H)$, there exists a valuation v such that $(\ell, v) \in \mathsf{Unsafe}([\![H]\!])$ and thus a trajectory π of $[\![H]\!] = \langle S, S_0, S_f, \Sigma, \to \rangle$ with $\mathsf{first}(\pi) \in S_0$, $\mathsf{last}(\pi) \in S_f$ and $\mathsf{state}_i(\pi) = (\ell, v)$ for some $i \in \mathbb{N}$. Since $[\![\mathsf{rect}(H)]\!] \succeq [\![H]\!]$ witnessed by the identity simulation relation, π is also a trajectory of $[\![\mathsf{rect}(H)]\!] = \langle S, S_0, S_f, \Sigma, \to' \rangle$. Therefore, $(\ell, v) \in \mathsf{Unsafe}([\![\mathsf{rect}(H)]\!])$ and $\ell \notin \mathsf{SafeLoc}(\mathsf{rect}(H))$. ■

7.3 Abstraction Refinement for Hybrid Automata

We explain how to refine a rectangular phase-portrait approximation of an hybrid automaton, that is how to obtain a more precise rectangular automaton, in the sense of simulation relations.

Definition 7.7 [Refined approximation] Given H' and H'' two rectangular phase-portrait approximations of an hybrid automaton H, we say that H'' *refines* H' if $[\![H']\!] \succeq [\![H'']\!]$. □

A natural way of refining an approximation of an affine automaton is to *split* its locations by partitioning or covering their invariant.

Definition 7.8 [Cut] Given a polytope $P \subseteq \mathbb{R}^n$ and a hyperplane $\pi \equiv f(x) = 0$, we define the *cut* $P/\pi = \langle P^+, P^- \rangle$ where $P^+ = P \cap f^{-1}(\mathbb{R}^{\geq 0})$ and $P^- = P \cap f^{-1}(\mathbb{R}^{\leq 0})$. The cut P/π is said *non-trivial* if $P^+ \neq \varnothing$ and $P^- \neq \varnothing$. □

Thus a non-trivial cut P/π of a polytope is a *cover* of P but not a partition since the two pieces P^+ and P^- are closed sets and they share the points in $P \cap \pi$.

Definition 7.9 [Location splitting] Given an hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$, one of its locations $\ell^\star \in \mathsf{Loc}$ and a hyperplane $\pi \equiv f_\pi(x) = 0$ in $\mathbb{R}^{|X|}$, the *splitting* of H by the hyperplane π in location $\ell^\star$ is the hybrid automaton $\mathsf{split}(H, \ell^\star, \pi) = \langle \mathsf{Loc}', \mathsf{Lab}', \mathsf{Edg}', X', \mathsf{Init}', \mathsf{Inv}', \mathsf{Flow}', \mathsf{Jump}', \mathsf{Final}' \rangle$ where:

- $\mathsf{Loc}' = \mathsf{Loc} \backslash \{\ell^\star\} \cup \{(\ell^\star, \varphi_1), (\ell^\star, \varphi_2)\}$ where $\varphi_1 \equiv \mathsf{Inv}(\ell^\star) \wedge f_\pi(x) \leq 0$ and $\varphi_2 \equiv \mathsf{Inv}(\ell^\star) \wedge f_\pi(x) \geq 0$. For $\ell' \in \mathsf{Loc}'$, let $loc(\ell') = \ell'$ if $\ell' \in \mathsf{Loc}$ and $loc(\ell') = \ell^\star$ otherwise;
- $\mathsf{Lab}' = \mathsf{Lab}$;
- $\mathsf{Edg}' = E_1 \cup E_2$ where $E_1 = \{(\ell, \sigma, \ell') \mid \ell, \ell' \in \mathsf{Loc}' \wedge (loc(\ell), \sigma, loc(\ell')) \in \mathsf{Edg}\}$ is the set of edges inherited from H and $E_2 = \{((\ell^\star, \varphi_1), \tau, (\ell^\star, \varphi_2)), ((\ell^\star, \varphi_2), \tau, (\ell^\star, \varphi_1))\}$ are silent edges between the two copies of the location $\ell^\star$;
- $X' = X$;
- $\mathsf{Init}'(\ell') = \mathsf{Init}(loc(\ell'))$ for each $\ell' \in \mathsf{Loc}'$;
- $\mathsf{Inv}'(\ell') = \mathsf{Inv}(\ell')$ for each $\ell' \in \mathsf{Loc} \backslash \{\ell^\star\}$ and $\mathsf{Inv}'(\ell^\star, \varphi) = \varphi$;
- $\mathsf{Flow}'(\ell') = \mathsf{Flow}(loc(\ell'))$ for each $\ell' \in \mathsf{Loc}'$;
- for every $e = (\ell, \sigma, \ell') \in E_1$ we have $\mathsf{Jump}'(e) = \mathsf{Jump}\big((loc(\ell), \sigma, loc(\ell'))\big)$, and for every $e \in E_2$ we have $\mathsf{Jump}'(e) = \mathit{stable}(X)$;
- $\mathsf{Final}'(\ell') = \mathsf{Final}(loc(\ell'))$ for each $\ell' \in \mathsf{Loc}'$.

□

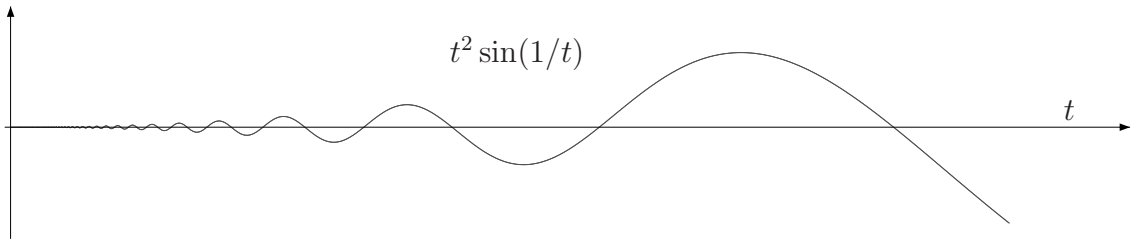


Figure 7.2: The function $f(t) = t^2 \sin(1/t)$.

Example Figure 7.1(b) shows for the shared gas-burner the rectangular phase-portrait approximation of the splitting of the location ℓ_1 by the line $x_1 = x_2$. The resulting automaton is a refinement since the ranges of the rectangular dynamics have decreased in each of the two splitted locations.

This technique is very general and has also been applied to hybrid automata with nonlinear dynamics [HHW98]. However, in that paper, the proof of correctness (that the refined automaton $\mathsf{split}(H, \ell^\star, \pi)$ weakly simulates the original automaton H) relies crucially on the fact that the split of an invariant is derived from a finite *open cover*, that is, a location $\ell^\star$ is replaced by $(\ell^\star, \varphi_1)$ and $(\ell^\star, \varphi_2)$ where $[\![\varphi_1]\!], [\![\varphi_2]\!]$ are open sets such that $\mathsf{Inv}(\ell^\star) \subseteq [\![\varphi_1]\!] \cup [\![\varphi_2]\!]$. Unfortunately, the proof cannot be extended to closed covers: for example, the continuously differentiable function $f : \mathbb{R}^{\geq 0} \to \mathbb{R}$ defined by $f(0) = 0$ and $f(t) = t^2 \sin(1/t)$ for $t > 0$, oscillates infinitely in every interval $[0, \epsilon]$ for $\epsilon > 0$, see Figure 7.2. So that if $f_{|[0,\delta]}$ was the witness of a transition $((\ell, v), \delta, (\ell, v'))$ of an automaton H with variable $X_H = \{y\}$, it would be impossible for the automaton $\mathsf{split}(H, \ell, y = 0)$ to mimic that transition since time cannot progress by any positive amount while maintaining either $y \geq 0$ or $y \leq 0$. However, this kind of pathological behaviour cannot occur as a solution of a system of affine dynamics flow conditions (even though spiral trajectories are still possible) because such solutions are *analytic* and zeroes of analytic functions are isolated (except for the identically zero function). Details are given in Appendix A.2.

Theorem 7.10 *For every affine hybrid automaton H, for every location ℓ of H, and every hyperplane π, we have $[\![\mathsf{split}(H, \ell, \pi)]\!] \approx_{weak} [\![H]\!]$, that is $[\![\mathsf{split}(H, \ell, \pi)]\!]$ and $[\![H]\!]$ are weakly bisimilar.*

With the remark above, the proof of Theorem 7.10 is straightforward.

We are interested in finding *automatically* an optimal cut for refining the state space. We consider the general problem to split a location in an optimal way, that is, to minimize the imprecision of the resulting rectangular phase-portrait approximation. The definition of the imprecision could have several forms. We decide to minimize the maximal size of the rectangular predicates that occur as flow conditions in the rectangular approximation of the splitted automaton (and particularly in the splitted

location). It may be necessary to scale the variables in order to give sense to the comparison of their dynamics range. We now discuss our choice. On the one hand, this criterion is natural since the precision of the approximation is directly connected to the size of the rectangular predicates. Further, minimizing the maximal size ensures that the approximation becomes more precise, and eventually arbitrarily precise. On the other hand, other criteria we have tried (minimizing the sum of the squares of the sizes, minimizing the size of the reachable set, etc.) gave rise to computational difficulties due to their non-linear form. Our criterion can be handled with linear programming techniques, and showed applicable in practice.

7.4 Optimization problems

We want to split the invariant of a location through a hyperplane, minimizing the maximal size of the rectangular predicates that approximate the affine dynamics. We define two versions of this problem, one called *concrete* when the given dynamics is *affine* on the invariant, and one called *abstract* when the invariant is already covered by a number of pieces each with its *rectangular* dynamics. The second formulation is introduced because we have no simple algorithm in $n\mathsf{D}$ for $n \geq 3$ for solving the concrete problem, while we have a general algorithm for the abstract one. Therefore, it may be of interest to discretize the original affine system and solve the abstract problem on the discretization. This yield an approximation of the optimal split for the concrete problem. We show that the resulting measure of imprecision can be made arbitrarily close to the exact solution. Let us define the two problems.

In Definition 7.11, we associate to each subset $Q \subseteq P$ of the invariant P of a location the tightest rectangular dynamics that contains the exact dynamics in the set Q (which is defined by an affine predicate of the form $\bigwedge_{x_i \in X} \dot{x_i} = f_i(x_1, \ldots, x_n)$). Then, the *imprecision* of a cut of P into two pieces is defined to be the maximal size of the rectangular predicates associated to each piece.

Definition 7.11 [Concrete imprecision] Let $X = \{x_1, \ldots, x_n\}$ be a finite set of variables. Let $P \subset \mathbb{R}^n$ be a polytope and $F = \langle f_1, \ldots, f_n \rangle$ be a tuple of n affine functions $f_i : \mathbb{R}^n \to \mathbb{R}$ $(1 \leq i \leq n)$. For a polytope $Q \subseteq P$, we define the rectangular predicate

$$\mathsf{range}^F(Q) = \bigwedge_{x \in X} \dot{x} \in I_x$$

where for each $x_i \in X$, we have $I_{x_i} = f_i(Q)$. We define the *concrete imprecision* of a cut $P/\pi = \langle P^+, P^- \rangle$ by:

$$\mathsf{sizeRange}^F(P/\pi) = \max\left\{|\mathsf{range}^F(P^+)|, |\mathsf{range}^F(P^-)|\right\}$$

□

Example We illustrate Definition 7.11 with the shared gas-burner, shown in Figure 2.1 and the rectangular approximations of Figure 7.1. We have $X = \{x_1, x_2\}$, $P = [0, 100] \times [0, 100]$ and $F = \langle h_1 - a_1 x_1 + b_1 x_2, -a_2 x_2 + b_2 x_1 \rangle$. The splitting of Figure 7.1(b) is obtained by the cut of P by the line $\pi \equiv x_1 = x_2$. Consider the location $\ell_1^{\geq}$ whose invariant is the triangle $Q^{\geq} = P \cap \{(x_1, x_2) \mid x_2 \geq x_1\}$. The rectangular dynamics in $\ell_1^{\geq}$ is $\mathsf{range}^F(Q^{\geq}) = \dot{x}_1 \in [\frac{3}{2}, \frac{5}{2}] \wedge \dot{x}_2 \in [-\frac{1}{2}, \frac{1}{2}]$, whose size is $|\mathsf{range}^F(Q^{\geq})| = 1$. Symmetrically, the rectangular dynamics in $\ell_1^{\leq}$ is $\mathsf{range}^F(Q^{\leq}) = \dot{x}_1 \in [1, 2] \wedge \dot{x}_2 \in [-1, 0]$ whose size is also 1. Hence, the concrete imprecision of the cut $\langle Q^{\geq}, Q^{\leq} \rangle$ is given by $\mathsf{sizeRange}^F(P/\pi) = \max\{1, 1\} = 1$.

We associate an *optimal-cut problem* with the concrete imprecision that asks to split P to minimize the imprecision.

Definition 7.12 [Concrete optimal-cut problem] An *instance of the concrete optimal-cut problem* is a tuple $\langle P, X, F \rangle$ where:

- $P \subset \mathbb{R}^n$ is a polytope,
- $X = \{x_1, \ldots, x_n\}$ is a set of n variables, and
- F is a tuple of n affine functions $f_i : \mathbb{R}^n \to \mathbb{R}$ $(1 \leq i \leq n)$.

Given $\langle P, X, F \rangle$, the *concrete optimal-cut problem* is to determine a hyperplane $\pi^\star \subset \mathbb{R}^n$ such that for every hyperplane $\pi \subset \mathbb{R}^n$, $\mathsf{sizeRange}^F(P/\pi^\star) \leq \mathsf{sizeRange}^F(P/\pi)$.

□

Observe that P is a polytope in Definition 7.12, which means that to apply the concrete optimal-cut problem to hybrid automata, we have to assume that the invariants are bounded.

We define an abstract version of the optimal-cut problem where the invariants of the locations are originally given as a union of polytopes $\mathcal{P}$, and the flow condition in each polytope is rectangular. This form of the problem is naturally obtained when we discretize the concrete problem according to $\mathcal{P}$. The abstract problem asks to split the location into only *two* pieces such that the maximal flow interval in the two pieces is minimized. This version of the optimal-cut problem can be used to approximate the optimal cut according to the concrete imprecision, as in Definition 7.12.

Definition 7.13 [Abstract imprecision] Let $X = \{x_1, \ldots, x_n\}$ be a finite set of variables. Let $P \subset \mathbb{R}^n$ be a polytope covered by a finite set of polytopes $\mathcal{P} = \{P_1, \ldots, P_m\}$, that is such that $P = P_1 \cup \cdots \cup P_m$. Let $\mathsf{Flow} : \mathcal{P} \to \mathsf{Rect}(\dot{X})$ be a function that associates to each polytope of $\mathcal{P}$ a rectangular dynamics. For a polytope $Q \subseteq P$, we define the rectangular predicate

$$\mathsf{range}^{\mathsf{Flow}}(Q) = \bigsqcup_{P_j \in \mathcal{P}, P_j \cap Q \neq \varnothing} \mathsf{Flow}(P_j)$$

We define the *abstract imprecision* of a cut $P/\pi = \langle P^+, P^- \rangle$ by:

$$\mathsf{sizeRange}^{\mathsf{Flow}}(P/\pi) = \max\left\{|\mathsf{range}^{\mathsf{Flow}}(P^+)|, |\mathsf{range}^{\mathsf{Flow}}(P^-)|\right\}$$

□

Definition 7.14 [Abstract optimal-cut problem] An *instance of the abstract optimal-cut problem* is a tuple $\langle P, \mathcal{P}, X, \mathsf{Flow} \rangle$ where:

- $P \subset \mathbb{R}^n$ is a polytope,
- $\mathcal{P} = \{P_1, \ldots, P_m\}$ is finite set of polytopes such that $P = P_1 \cup \cdots \cup P_m$,
- $X = \{x_1, \ldots, x_n\}$ is a set of n variables, and
- $\mathsf{Flow} : \mathcal{P} \to \mathsf{Rect}(\dot{X})$.

Given $\langle P, \mathcal{P}, X, \mathsf{Flow} \rangle$, the *abstract optimal-cut problem* asks to determine a hyperplane $\pi^\star \subset \mathbb{R}^n$ such that $\mathsf{sizeRange}^{\mathsf{Flow}}(P/\pi^\star) \leq \mathsf{sizeRange}^{\mathsf{Flow}}(P/\pi)$ for every hyperplane $\pi \subset \mathbb{R}^n$. □

7.4.1 Solution of the abstract optimal-cut problem

We give an algorithm for solving the abstract problem and show how it can be used to approximate the solution of the concrete problem.

Definition 7.15 [Separability] Two sets $A, B \subseteq \mathbb{R}^n$ are *separable* if there exists an affine function $f : \mathbb{R}^n \to \mathbb{R}$ such that $\forall x \in A : f(x) \leq 0$ and $\forall x \in B : f(x) \geq 0$. □

This definition extends to sets of sets: we say that $\mathcal{A} \subseteq 2^{\mathbb{R}^n}$ and $\mathcal{B} \subseteq 2^{\mathbb{R}^n}$ are separable if $\bigcup \mathcal{A}$ and $\bigcup \mathcal{B}$ are separable. Separability can be tested using the *convex hull* of a set, denoted $\mathsf{Conv}(\cdot)$.

Lemma 7.16 *Two sets $A, B \subseteq \mathbb{R}^n$ are separable iff there exists a hyperplane $\pi \subset \mathbb{R}^n$ such that $\mathsf{Conv}(A) \cap \mathsf{Conv}(B) \subseteq \pi$.*

An optimal cut for the abstract problem $\langle P, \mathcal{P}, X, \mathsf{Flow} \rangle$ is computed by Algorithm 5. It uses two external functions *value* and *separable* taking as input two sets $\mathcal{A}$ and $\mathcal{B}$ of polytopes. The function *value* returns the number $\mathsf{sizeRange}^{\mathsf{Flow}}(\langle \bigcup \mathcal{A}, \bigcup \mathcal{B} \rangle)$ and the boolean function *separable* returns true iff $\mathcal{A}$ and $\mathcal{B}$ are separable. Lemma 7.16 suggests a natural implementation of *separable*. Notice that constructing a hyperplane that separates two polytopes given by their constraints representation is a linear program.

The principle of Algorithm 5 is the following: it constructs incrementally two sets of pieces $G^+ \subseteq \mathcal{P}$ and $G^- \subseteq \mathcal{P}$ to separate, and maintains a set $G_0 = \mathcal{P} \backslash (G^+ \cup G^-)$ of untreated pieces. Initially, we have $G^+ = G^- = \varnothing$ and $G_0 = \mathcal{P}$. The call $split(\varnothing, \varnothing, \mathcal{P})$ is aimed to return two sets G^+ and G^- that are separable and such that any separating hyperplane of G^- and G^+ is an optimal cut for $\langle P, \mathcal{P}, X, \mathsf{Flow} \rangle$. Intuitively, the function *split* iteratively selects and separates two pieces $P_i, P_j \in \mathcal{P}$ with a maximal flow interval $r = \mathsf{range}^{\mathsf{Flow}}(P_i \cup P_j)$, so that if P_i and P_j were not separated there would be no way that the final imprecision run below r. The constraint that P_i and P_j must be separated by the optimal cut can be represented by putting an edge between P_i and P_j in a graph whose vertices is the set $\mathcal{P}$. We can add new edges as long as the graph remains 2-colorable *and* the two sets of pieces induced by the 2-coloring is physically separable by a hyperplane. In the case the new edge is already connected to the rest of the graph, the color of the common vertex imposes the color of the other. Otherwise, the algorithm has to explore two choices (corresponding to put either P_i in G^- and P_j in G^+ or vice versa). An obvious argument shows that this could occur at most n times (where $n = |X|$ is the number of variables) so that the the algorithm is in $\mathrm{O}(m.2^n)$ (with $m = |\mathcal{P}|$), assuming constant execution time of external functions *value* and *separable*. We do not know if this bound is tight for the problem.

Theorem 7.17 *Algorithm 5 is correct and terminates.*

Proof. We prove the correctness of the function *split* by showing that if the pre-condition Pre below is satisfied, then we have the post-condition Post for the assignment $\langle D, E, v \rangle \leftarrow split(A, B, C)$:

- $\mathsf{Pre} \equiv A \cup B \cup C = \mathcal{P} \wedge A$ and B are separable.
- $\mathsf{Post} \equiv A \subseteq D \subseteq \mathcal{P} \wedge B \subseteq E \subseteq \mathcal{P} \wedge$ any cut separating D and E is the best among those separating A and B (and its value is v).

Obviously, the pre-condition is satisfied by the call $split(\varnothing, \varnothing, \mathcal{P})$, and the post-condition ensures that G^- and G^+ are separable by the optimal cut.

The proof of correctness is by induction on the size of the third argument $|C|$. First, we consider the case $|C| = 0$. Second, we show that the correctness for $|C| \leq n - 1$ implies the correctness for $|C| = n$. Finally, we note that whenever $split(A, B, C)$ calls $split(A', B', C')$, we have $|C'| < |C|$ which allows to conclude that the algorithm is correct and always terminates.

1. Assume $|C| = 0$. Then $C = \varnothing$, and the test of line 2 is satisfied. The call $split(A, B, C)$ returns $\langle A, B, value(A, B) \rangle$ which trivially satisfies the post-condition.

Algorithm 5: Algorithm for the abstract optimal-cut problem.

Input : An instance $\langle P, \mathcal{P}, X, \mathsf{Flow}\rangle$ of the abstract optimal-cut problem.

Result : Two separable sets G^- and G^+ such that any separating hyperplane of G^- and G^+ is an optimal cut for $\langle P, \mathcal{P}, X, \mathsf{Flow}\rangle$.

```
begin
    return split(∅, ∅, 𝒫) ;
end

external function value(𝒜,ℬ: set of polytopes): ℝ^{≥0}

external function separable(𝒜,ℬ: set of polytopes): {true, false}

function split(G⁻, G⁺, G₀: set of polytopes): 2^𝒫 × 2^𝒫 × ℝ^{≥0} begin
1   Let P_i, P_j ∈ 𝒫 maximizing range^Flow(P_i ∪ P_j) subject to P_i ∈ G₀ ∨ P_j ∈ G₀ ;
2   if no such P_i, P_j exists then return ⟨G⁻, G⁺, value(G⁻ ∪ G₀, G⁺ ∪ G₀)⟩ ;
3   if P_i ∈ G₀ ∧ P_j ∈ G₀ then
4       v_A ← ∞ ;
5       v_B ← ∞ ;
6       if separable(G⁻ ∪ {P_i}, G⁺ ∪ {P_j}) then
7           ⟨A₁, A₂, v_A⟩ ← split(G⁻ ∪ {P_i}, G⁺ ∪ {P_j}, G₀\{P_i, P_j}) ;
8       if separable(G⁻ ∪ {P_j}, G⁺ ∪ {P_i}) then
9           ⟨B₁, B₂, v_B⟩ ← split(G⁻ ∪ {P_j}, G⁺ ∪ {P_i}, G₀\{P_i, P_j}) ;
10      if v_A = v_B = ∞ then return ⟨G⁻, G⁺, value(G⁻ ∪ G₀, G⁺ ∪ G₀)⟩ ;
11      if v_A ≤ v_B then
12          return ⟨A₁, A₂, v_A⟩ ;
        else
13          return ⟨B₁, B₂, v_B⟩ ;
    else
14      Assume w.l.o.g. that P_i ∈ G⁻ ;
15      if separable(G⁻, G⁺ ∪ {P_j}) then
16          return split(G⁻, G⁺ ∪ {P_j}, G₀\{P_j}) ;
        else
17          return ⟨G⁻, G⁺, value(G⁻ ∪ G₀, G⁺ ∪ G₀)⟩ ;
end
```

2. Let $n > 0$. Assume $split(A, B, C)$ is correct for $|C| \leq n - 1$. Consider the call $split(A, B, C)$ with $|C| = n$. Then $C \neq \varnothing$. Hence, some P_i and P_j exist and the test of line 2 fails. We claim that among the cuts separating A and B, any cut separating P_i and P_j is better than any other cut. This is because at line 1, we have considered every pair of polytopes in $\mathcal{P}$ for which we do not already know whether or not they must be separated. Therefore, if it is possible to separate both A and B and P_i and P_j (that is separate either $A \cup \{P_i\}$ and $B \cup \{P_j\}$ or $A \cup \{P_j\}$ and $B \cup \{P_i\}$), we must do so, and otherwise, every cut separating A and B is optimal (and such a cut exists by the pre-condition).

   (a) If the condition of line 3 evaluates to true. Then, we compare the cost of separating $A \cup \{P_i\}$ and $B \cup \{P_j\}$ or vice versa, and take the best. If both choices lead to inseparable sets, we have $v_A = v_B = \infty$ and the function returns $\langle A, B\rangle$. Notice that the pre-conditions for the recursive calls are trivially satisfied.

   (b) Otherwise, one of P_i and P_j is already in either A or B. Assume w.l.o.g. that $P_i \in A$. Then, the only way to possibly separate P_i and P_j is to put P_j in B. The test of line 15 checks this and the function returns the best cut in each case.

■

Algorithm 5 can be used to solve the concrete optimal-cut problem up to any precision $\epsilon \in \mathbb{Q}^{>0}$. It suffices to discretize the given polytope with a grid of size ϵ: given a tuple $F = \langle f_1, \ldots, f_n\rangle$ of n affine functions in $\mathbb{R}^n$, let

$$\mathsf{Grid}_\epsilon^F = \{ \bigcap_{1 \leq i \leq n} f_i^{-1}([k_i\,\epsilon, (k_i + 1)\,\epsilon]) \mid (k_1, \ldots, k_n) \in \mathbb{Z}^n \}$$

In practice, the complexity blows up since the number of elements in the grid increases exponentially with $1/\epsilon$.

Definition 7.18 [ϵ-discretization of a concrete optimal-cut problem] Let $Q = \langle P, X, F\rangle$ be an instance of the concrete optimal-cut problem in $\mathbb{R}^n$, and let $\epsilon \in \mathbb{Q}^{>0}$. The *$\epsilon$-discretization* of Q is the instance $Q_\epsilon = \langle P, \mathcal{P}, X, \mathsf{Flow}\rangle$ of the abstract optimal-cut problem such that:

- $\mathcal{P} = \{P \cap box \mid box \in \mathsf{Grid}_\epsilon^F \wedge P \cap box \neq \varnothing\}$. Notice that $\mathcal{P}$ is finite since P is bounded;
- for each $P_j \in \mathcal{P}$, we have $\mathsf{Flow}(P_j) = \mathsf{range}^F(P_j)$ which is a rectangular predicate of size at most ϵ.

□

Theorem 7.19 *Let $Q = \langle P, X, F\rangle$ be an instance of the concrete optimal-cut problem and let $Q_\epsilon = \langle P, \mathcal{P}, X, \mathsf{Flow}\rangle$ be its ϵ-discretization for some $\epsilon \in \mathbb{Q}^{>0}$. If $\pi^\star$ is a solution for Q and $\pi^\star_\epsilon$ is a solution for Q_ϵ, then $0 \leq \mathsf{sizeRange}^F(P/\pi^\star_\epsilon) - \mathsf{sizeRange}^F(P/\pi^\star) < \epsilon$.*

Proof. Let $\epsilon \in \mathbb{Q}^{>0}$. Let $P/\pi^\star = \langle P^+, P^-\rangle$ and $v^\star = \mathsf{sizeRange}^F(P/\pi^\star)$. For each f_i in F, let $I_i = f_i(P)$. First, we claim that for all $1 \leq i \leq n$ such that $\mathsf{size}(I_i) \geq v^\star$, the hyperplane $\pi^\star$ separates the sets

$$\mathsf{Left}_i = \{x \in P \mid f_i(x) \leq r(I_i) - v^\star\} \text{ and } \mathsf{Right}_i = \{x \in P \mid f_i(x) \geq l(I_i) + v^\star\}$$

This follows from the fact that for all $x \in \mathsf{Left}_i$ and $x' \in \mathsf{Right}_i$, the hyperplane $\pi^\star$ separates x and x'. To show this, let $x_\mathsf{L}, x_\mathsf{R} \in P$ such that $f_i(x_\mathsf{L}) = l(I_i)$ and $f_i(x_\mathsf{R}) = r(I_i)$. The following points are separated by $\pi^\star$: (i) x and x_R (ii) x_R and x_L (iii) x_L and x'. Therefore, x and x' are separated by $\pi^\star$.

Now, for $a \in \mathbb{R}$, let $\lfloor a \rfloor$ (resp. $\lceil a \rceil$) be the greatest (resp. lowest) integer k such that $k \leq a$ (resp. $k \geq a$) and let $\lfloor a \rfloor_\epsilon = \epsilon \lfloor \frac{a}{\epsilon} \rfloor$ and $\lceil a \rceil_\epsilon = \epsilon \lceil \frac{a}{\epsilon} \rceil$. Let

$$\mathsf{Left}^\epsilon_i = \{x \in P \mid f_i(x) \leq \lfloor r(I_i) - v^\star \rfloor_\epsilon\} \text{ and } \mathsf{Right}^\epsilon_i = \{x \in P \mid f_i(x) \geq \lceil l(I_i) + v^\star \rceil_\epsilon\}$$

Clearly, we have the inclusions $\mathsf{Left}^\epsilon_i \subseteq \mathsf{Left}_i$ and $\mathsf{Right}^\epsilon_i \subseteq \mathsf{Right}_i$ so that $\pi^\star$ also separates Left^ϵ_i and $\mathsf{Right}^\epsilon_i$. Thus we can assume that $\mathsf{Left}^\epsilon_i \subseteq P^+$ and $\mathsf{Right}^\epsilon_i \subseteq P^-$. Therefore, we have:

$$f_i(P^+) \subseteq \left[l(I_i), \lceil l(I_i) + v^\star \rceil_\epsilon\right] \text{ and } f_i(P^-) \subseteq \left[\lfloor r(I_i) - v^\star \rfloor_\epsilon, r(I_i)\right]$$

Since for all $a \in \mathbb{R}$ we have $\lfloor a \rfloor_\epsilon > a - \epsilon$ and $\lceil a \rceil_\epsilon < a + \epsilon$, we conclude that $\mathsf{sizeRange}^{\mathsf{Flow}}(P/\pi^\star) < v^\star + \epsilon$. It is easy to show that:

$$\mathsf{sizeRange}^F(P/\pi^\star_\epsilon) \leq \mathsf{sizeRange}^{\mathsf{Flow}}(P/\pi^\star_\epsilon) \leq \mathsf{sizeRange}^{\mathsf{Flow}}(P/\pi^\star)$$

which yields $\mathsf{sizeRange}^F(P/\pi^\star_\epsilon) < v^\star + \epsilon$.

■

7.4.2 Solution of the concrete optimal-cut problem in $\mathbb{R}^2$

We propose Algorithm 6 to solve the concrete optimal-cut problem $\langle P, X, F\rangle$ in two dimensions ($P \subset \mathbb{R}^2$), when $F = \langle f_1, f_2\rangle$ contains two functions. This algorithm is inspired by the abstract Algorithm 5 applied to an ϵ-discretization of the concrete problem with $\epsilon \to 0$. The main trick is to translate the condition of separability expressed with convex hulls into a more continuous condition. We show that this condition can be computed by a linear program.

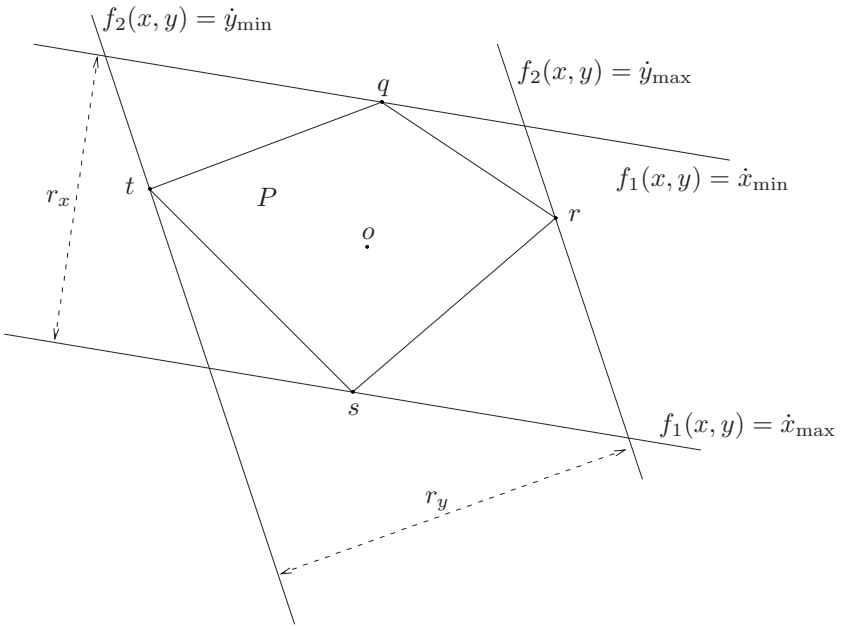


Figure 7.3: A polytope in $\mathbb{R}^2$ with 2 affine functions f_1 and f_2.

Let us execute Algorithm 6 and explain informally why it is correct. The input is an instance $\langle P, X, \{f_1, f_2\}\rangle$ of the concrete optimal-cut problem. We represent P on Figure 7.3.

At lines 1,2 we compute the interval image of P by f_1 and f_2, and the size of the those intervals r_x and r_y. The assumption $r_y \geq r_x$ of line 3 implies that the points r and t on Figure 7.3 are such that $\mathsf{sizeRange}^F(\{r,t\}) = \mathsf{sizeRange}^F(P)$, that is r and t realize the maximal imprecision on P. Therefore, any cut that separates those points is better than any other cut. This remains true for all pairs of points taken in the shaded regions defined by Δ_0 on Figure 7.4(a) until:

- either Δ_0 becomes equal to $r_y - r_x$, which means that separating two points q and s is "as necessary" as separating the shaded regions;
- or Δ_0 reaches the value $\frac{r_y}{2}$, and the optimal-cut is given by the line $\ell \equiv f_2(x,y) = \dot{y}_{\min} + \frac{r_y}{2}$.

This alternative is tested at line 4: the condition $r_y \geq 2\,r_x$ is equivalent to $\frac{r_y}{2} \leq r_y - r_x$. If $r_y < 2\,r_x$, the algorithm continues as depicted on Figure 7.4(b), separating all pairs of points that give the largest range for function f_1 and f_2. The four regions P_q, P_r, P_s and P_t (containing respectively the points q, r, s and t) are growing altogether at the same "rate". The algorithm will stop whenever it becomes impossible to separate both P_q from P_s and P_r from P_t. As in the abstract algorithm, there are two branches

Algorithm 6: Algorithm for computing the concrete optimal-cut in 2D.

Input : An instance $S = \langle P, X, F\rangle$ of the concrete optimal-cut problem with $P \subset \mathbb{R}^2$, $X = \{x, y\}$, and $F = \langle f_1, f_2\rangle$.

Result : A line that solves the concrete optimal-cut problem for S.

begin

$[\dot{x}_{\min}, \dot{x}_{\max}] \leftarrow f_1(P)$; $[\dot{y}_{\min}, \dot{y}_{\max}] \leftarrow f_2(P)$;

$r_x \leftarrow \dot{x}_{\max} - \dot{x}_{\min}$; $r_y \leftarrow \dot{y}_{\max} - \dot{y}_{\min}$;

Assume w.l.o.g. that $r_y \geq r_x$;

if $r_y \geq 2\, r_x$ **then return** $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

$\Delta_0 \leftarrow r_y - r_x$;

Let Δ be a symbolic parameter ;

$a_\Delta \leftarrow f_2^{-1}(\dot{y}_{\min} + \Delta_0 + \Delta) \cap f_1^{-1}(\dot{x}_{\min} + \Delta)$;

$b_\Delta \leftarrow f_1^{-1}(\dot{x}_{\min} + \Delta) \cap f_2^{-1}(\dot{y}_{\max} - \Delta_0 - \Delta)$;

$c_\Delta \leftarrow f_2^{-1}(\dot{y}_{\max} - \Delta_0 - \Delta) \cap f_1^{-1}(\dot{x}_{\max} - \Delta)$;

$d_\Delta \leftarrow f_1^{-1}(\dot{x}_{\max} - \Delta) \cap f_2^{-1}(\dot{y}_{\min} + \Delta_0 + \Delta)$;

for $z = a$ ***to*** d **do** $\Delta_z \leftarrow \min\{\Delta \mid z_\Delta \in P\}$;

$\Delta_1 \leftarrow \min(\Delta_a, \Delta_c)$; $\Delta_2 \leftarrow \min(\Delta_b, \Delta_d)$;

if $\Delta_1 \geq \frac{r_y}{2} - \Delta_0 \ \vee \ \Delta_2 \geq \frac{r_y}{2} - \Delta_0$ **then return** $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

$Q_{\min} \leftarrow P \cap f_1^{-1}(\dot{x}_{\min})$; $Q_{\max} \leftarrow P \cap f_1^{-1}(\dot{x}_{\max})$;

if $f_2(Q_{\min}) \cap [\dot{y}_{\min}, \dot{y}_{\min} + \Delta_0] \neq \varnothing \ \wedge \ f_2(Q_{\min}) \cap [\dot{y}_{\max} - \Delta_0, \dot{y}_{\max}] \neq \varnothing$ **then**

  return $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

else if $f_2(Q_{\min}) \cap [\dot{y}_{\min}, \dot{y}_{\min} + \Delta_0] \neq \varnothing$ **then**

  if $f_2(Q_{\max}) \cap [\dot{y}_{\min}, \dot{y}_{\min} + \Delta_0] \neq \varnothing$ **then**

    return $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

  else

    return $\ell ine(b_{\Delta_2}, d_{\Delta_2})$;

else if $f_2(Q_{\min}) \cap [\dot{y}_{\max} - \Delta_0, \dot{y}_{\max}] \neq \varnothing$ **then**

  if $f_2(Q_{\max}) \cap [\dot{y}_{\max} - \Delta_0, \dot{y}_{\max}] \neq \varnothing$ **then**

    return $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

  else

    return $\ell ine(a_{\Delta_1}, c_{\Delta_1})$;

else if $f_2(Q_{\max}) \cap [\dot{y}_{\max} - \Delta_0, \dot{y}_{\max}] \neq \varnothing \ \wedge \ f_2(Q_{\max}) \cap [\dot{y}_{\min}, \dot{y}_{\min} + \Delta_0] \neq \varnothing$ **then**

  return $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$;

else if $f_2(Q_{\max}) \cap [\dot{y}_{\max} - \Delta_0, \dot{y}_{\max}] \neq \varnothing$ **then**

  return $\ell ine(b_{\Delta_2}, d_{\Delta_2})$;

else if $f_2(Q_{\max}) \cap [\dot{y}_{\min}, \dot{y}_{\min} + \Delta_0] \neq \varnothing$ **then**

  return $\ell ine(a_{\Delta_1}, c_{\Delta_1})$;

else if $\Delta_1 > \Delta_2$ **then**

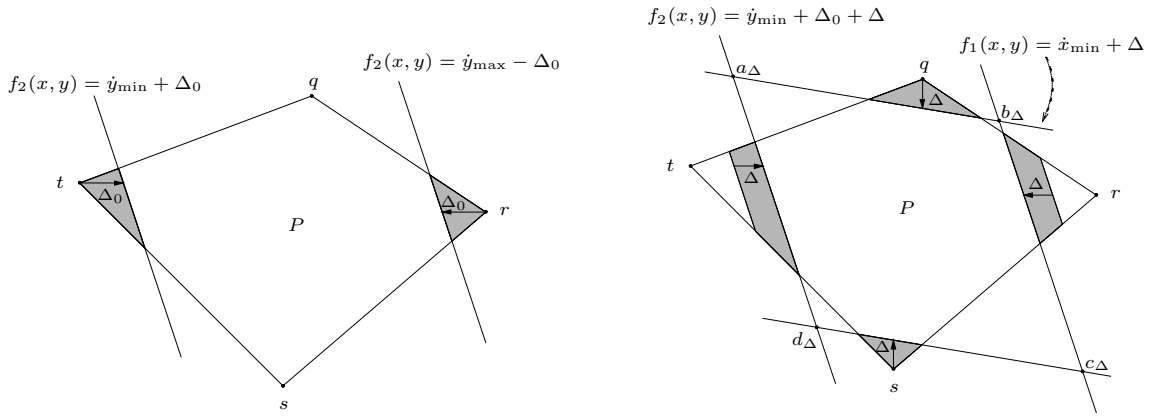
  return $\ell ine(a_{\Delta_1}, c_{\Delta_1})$;

else

  return $\ell ine(b_{\Delta_2}, d_{\Delta_2})$;

end

to explore, corresponding to separate either $\{P_q, P_r\}$ from $\{P_s, P_t\}$ or $\{P_q, P_t\}$ from $\{P_r, P_s\}$. Consider the points $a_\Delta, b_\Delta, c_\Delta$ and d_Δ that are the intersections of the level sets of f_1 as sketched on Figure 7.4(b). The subscript emphasizes the fact that those points are moving when Δ varies.



(a) First phase.

(b) Second Phase.

Figure 7.4: Two phases of Algorithm 6.

Intuitively, the sets $\{P_q, P_r\}$ and $\{P_s, P_t\}$ are separable iff a_Δ and c_Δ are outside P, a possible separating line being the line connecting a_Δ and c_Δ. Similarly, $\{P_q, P_t\}$ and $\{P_r, P_s\}$ are separable iff b_Δ and d_Δ are outside P. Assume that, as Δ increases, one of the points b_Δ or d_Δ first enters P. Then, it becomes impossible to separate $\{P_q, P_t\}$ from $\{P_r, P_s\}$. But since $\{P_q, P_r\}$ and $\{P_s, P_t\}$ are still separable (a_Δ and c_Δ are outside P), the algorithm can continue increase Δ. When either a_Δ or c_Δ enters P, the algorithm stops (with say $\Delta = \Delta^\star$). An optimal line cut is given by the line passing by $a_{\Delta^\star}$ and $c_{\Delta^\star}$. Several other particular configurations may occur, depending on the relative position of the level sets of f_1 and f_2 and the polytope P. Those cases are treated in details in the proof of correctness of the algorithm (see for example Figure 7.9).

In the algorithm, the points a_Δ, b_Δ, c_Δ and d_Δ are defined at lines 7–10. Then for $z = a \ldots d$, we compute the value Δ_z such that z_Δ "enters" P as a linear program: it corresponds to the minimal value of Δ such that $z_\Delta \in P$. From the intuitive interpretation above, it appears that $\Delta^\star = \max(\min(\Delta_a, \Delta_c), \min(\Delta_b, \Delta_d))$. The many particular cases are treated by the algorithm from line 13. Finally, the optimal cut in the general case is returned according to the test of line 31 (as in the example).

Remarks (*i*) The macro-instructions of Algorithm 6 (lines 1, 11, 14) can be implemented by linear programs. In particular, the coordinates of a_Δ, b_Δ, c_Δ and d_Δ are

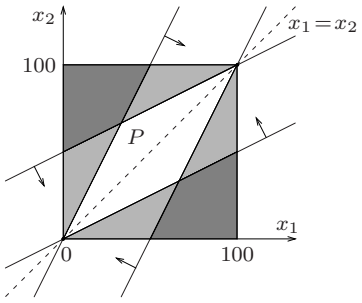


Figure 7.5: The optimal cut of the invariant of location ℓ_1 of the shared gas-burner.

linear in the parameter Δ so that the minimization of line 11 is indeed a linear program. (*ii*) A property of the line cut returned by Algorithm 6 is that it passes by the center of the parallelogram $qrst$ defined by the extremal level sets of f_1 and f_2 on P (the point o on Figure 7.3). We do not need this property so we omit its proof.

Example Figure 7.5 shows the invariant of location ℓ_1 of the shared gas-burner of Figure 2.1, with the position of the level sets of $f_1(x_1, x_2) = h_1 - a_1x_1 + b_1x_2$ and $f_2(x_1, x_2) = -a_2x_2 + b_2x_1$ at the end of Algorithm 6. The four arrows indicates the moving direction of the lines corresponding to increase the parameter Δ in the algorithm. The shaded regions corresponds to the pieces that are to be separated. The optimal cut is the dashed line $x_1 = x_2$.

To establish the correctness of Algorithm 6, we need some preliminary definitions and lemmas.

First, we complete Definition 7.11 with the following: for a finite set $X = \{x_1, \ldots, x_n\}$ of n variables, a polytope $Q \subset \mathbb{R}^n$ and a tuple $F = \langle f_1, \ldots, f_n \rangle$ of n affine functions $f_i : \mathbb{R}^n \to \mathbb{R}$ $(1 \leq i \leq n)$, we have defined $\mathsf{range}^F(Q) = \bigwedge_{x \in X} \dot{x} \in I_x$ where for each $x_i \in X$, we have $I_{x_i} = f_i(Q)$. For each variable $x_i \in X$, we define $\mathsf{sizeRange}^F_{x_i}(Q) = \mathsf{size}(I_{x_i})$ the imprecision on $\dot{x}_i$ in Q.

Lemma 7.20 *Let $\langle P, X, F \rangle$ be an instance of the optimal-cut problem for which a solution is $\pi^\star$. Then* $\mathsf{sizeRange}^F(P/\pi^\star) \geq \frac{1}{2}\mathsf{sizeRange}^F(P)$.

Proof. Assume without loss of generality that $\mathsf{sizeRange}^F(P) = \mathsf{sizeRange}^F_{x_1}(P)$, and let $f_1(P) = [a, b]$. For every hyperplane π, let $P/\pi = \langle P^+, P^- \rangle$. We have $P^+ \cup P^- = P$ and thus $f_1(P^+) \cup f_1(P^-) = f_1(P)$. Therefore,

$$\mathsf{sizeRange}^F_{x_1}(P^+) + \mathsf{sizeRange}^F_{x_1}(P^-) \geq \mathsf{sizeRange}^F_{x_1}(P)$$

$$\text{so that } \max\left\{\mathsf{sizeRange}^F_{x_1}(P^+), \mathsf{sizeRange}^F_{x_1}(P^-)\right\} \geq \frac{1}{2}\mathsf{sizeRange}^F_{x_1}(P)$$

which entails the result since $\mathsf{sizeRange}^F(P/\pi^\star) = \max\left\{\mathsf{sizeRange}^F_{x_i}(P^\sim) \mid 1 \leq i \leq n \wedge \sim \in \{+,-\}\right\}$. ■

Lemma 7.21 *Let $f_1, f_2 : \mathbb{R}^2 \to \mathbb{R}$ be two affine functions that are not constant and not parallel. Let $a, b \in \mathbb{R}$. The set $\ell_1 = \bigcup_{t\in\mathbb{R}} f_1^{-1}(a+t) \cap f_2^{-1}(b+t)$ is a line*[1].

Lemma 7.22 *Let $C \subseteq \mathbb{R}^2$ be a convex set and $f : \mathbb{R}^2 \to \mathbb{R}$ an affine function such that $C \cap f^{-1}(0) = \varnothing$. If $C \cap f^{-1}(\mathbb{R}^{>0}) \neq \varnothing$, then $C \cap f^{-1}(\mathbb{R}^{<0}) = \varnothing$ (and vice versa, switching $\mathbb{R}^{>0}$ and $\mathbb{R}^{<0}$).*

Proof. Let $u \in C \cap f^{-1}(\mathbb{R}^{>0})$. Assume (*ad absurdum*) that there exist $v \in C \cap f^{-1}(\mathbb{R}^{<0})$. Then, for every $\lambda \in [0,1]$, $\lambda\, u + (1-\lambda)\, v \in C$. Consider the function $g(\lambda) = f(\lambda\, u + (1-\lambda)\, v)$. Obviously, g is continuous, $g(0) < 0$ and $g(1) > 0$. Thus by the intermediate value theorem, there exists some $\lambda^\star \in [0,1]$ such that $g(\lambda^\star) = 0$, that is, $f(w) = 0$ for $w = \lambda^\star\, u + (1-\lambda^\star)\, v$. Since C is convex, $w \in C$ thus a contradiction with the hypothesis $C \cap f^{-1}(0) = \varnothing$. ■

Lemma 7.23 *Let $\Pi = abcd$ be a parallelogram with center o (that is, $[a,c] \cap [b,d] = \{o\}$). Let C be a convex set and q, r, s, t be four points on the sides of Π that belong to C. More precisely, $q \in C \cap [a,b]$, $r \in C \cap [b,c]$, $s \in C \cap [c,d]$ and $t \in C \cap [d,a]$. Then $o \in C$.*

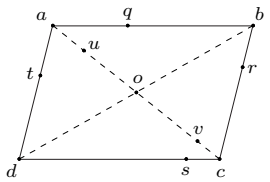


Figure 7.6: Illustration of Lemma 7.23.

Proof. It is easy to show the existence of $u \in [o,a]$ and $v \in [o,c]$ such that $u = \lambda q + (1-\lambda)t$ and $v = \mu r + (1-\mu)s$ for some $\lambda, \mu \in [0,1]$. Then, obviously o is expressible as a convex combination of u and v, and thus of q, r, s and t, which implies that $o \in C$ since C is convex (see also Figure 7.6). ■

[1]Notice that we can replace t by $-t$ in this definition without changing the set ℓ_1, since t is quantified over $\mathbb{R}$.

Lemma 7.24 *Let $\Pi = abcd$ be a parallelogram. Let $q \in [a,b]$, $r \in [b,c]$, $s \in [c,d]$, $t \in [a,d]$. If C is a convex set containing r and t, then $C \cap [q,s] \neq \varnothing$.*

Lemma 7.25 *Let $A, B \subseteq \mathbb{R}^2$ be two sets such that $\text{int}(A \cap B) \neq \varnothing$. Then A and B are not separable.*

Proof. Since $\text{int}(A \cap B) \neq \varnothing$, there exist $x \in A \cap B$ and $\epsilon > 0$ such that $B_\epsilon(x) \subseteq A \cap B$ (where $B_\epsilon(x) = \{y \in \mathbb{R}^2 \mid d(x,y) \leq \epsilon\}$ is the ball of radius ϵ centered in x and d is the euclidean distance. Hence, there are three points $x_1, x_2, x_3 \in A \cap B$ that do not lie on the same line. But, if there existed an affine function f defining a line cut separating A and B, the three numbers $f(x_1), f(x_2), f(x_3)$ would be equal to zero and so x_1, x_2 and x_3 would be aligned, which is a contradiction according to Lemma 7.16. ■

Definition 7.26 [Degenerated polytope] A polytope $P \subseteq \mathbb{R}^2$ is said *degenerated* iff there exists a line ℓ such that $P \subseteq \ell$. □

Lemma 7.27 *A polytope is degenerated iff its interior is empty.*

Lemma 7.28 *Let P be a non-degenerated polytope and $\Pi = abcd$ be a non-degenerated parallelogram such that $[a,c] \subseteq P$. Then $\text{int}(P \cap \Pi) \neq \varnothing$.*

Correctness of Algorithm 6

Theorem 7.29 *Algorithm 6 is correct and always terminates.*

Proof. The input of the algorithm is an instance $\langle P, X, F\rangle$ of the concrete optimal-cut problem, with $P \subset \mathbb{R}^2$, $X = \{x, y\}$ and $F = \langle f_1, f_2\rangle$. After lines 1-3 have executed, consider the case $r_y \geq 2\, r_x$. Then the algorithm stops at line 4, returning the line $\ell \equiv f_2(x,y) = \dot{y}_{\min} + \frac{r_y}{2}$. Let $P/\ell = \langle P^+, P^-\rangle$. It is clear that:

$$\begin{array}{ll} \mathsf{sizeRange}_x^F(P^+) \leq r_x & \mathsf{sizeRange}_x^F(P^-) \leq r_x \\ \mathsf{sizeRange}_y^F(P^+) = \frac{r_y}{2} & \mathsf{sizeRange}_y^F(P^-) = \frac{r_y}{2} \end{array}$$

Hence, $\mathsf{sizeRange}^F(P/\ell) = \frac{r_y}{2} = \frac{1}{2}\mathsf{sizeRange}^F(P)$. From Lemma 7.20, there exists no better cut, so ℓ must be optimal. Now, we assume that :

$$r_y < 2\, r_x \tag{7.1}$$

First, we show the existence of Δ_a, Δ_b, Δ_c and Δ_d computed at line 11, and thus the existence of the line cut returned by the algorithm. Second, we show that this cut is optimal. Let

$$\Delta_0 = r_y - r_x \text{ as in the algorithm, line 5.} \tag{7.2}$$

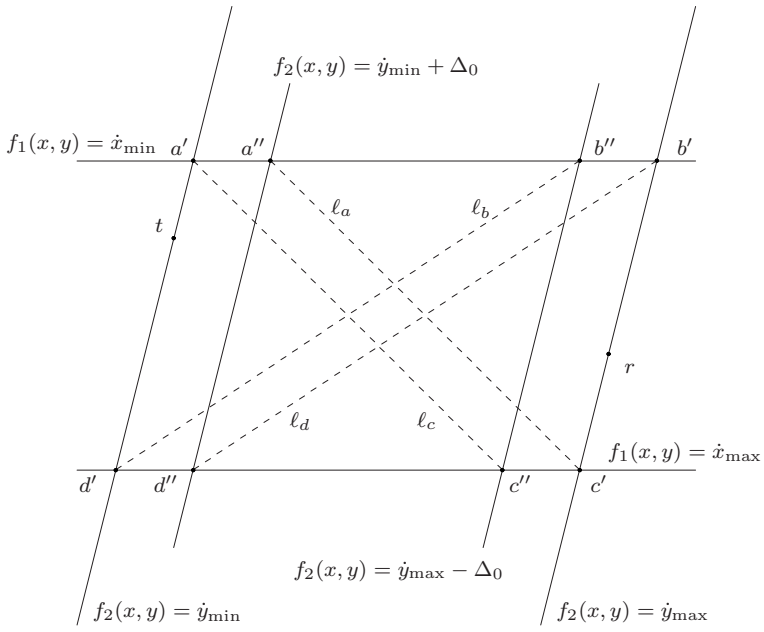


Figure 7.7: Level sets of f_1 and f_2 after the first phase of the algorithm.

1. To prove the existence of Δ_a, we show that the set $S_a = \{\Delta \mid a_\Delta \in P\}$ is nonempty, with the help of Figure 7.7. From Lemma 7.21, the set $\ell_a = \{a_\Delta \mid \Delta \in \mathbb{R}\}$ is a line and since $a_0 = a''$ and $a_{r_y - \Delta_0} = a_{r_x} = c'$, we have $\ell_a = \ell ine(a'', c')$. See Figure 7.7. If we apply Lemma 7.24 with the parallelogram $\Pi = a'b'c'd'$ defined by the level sets $f_1^{-1}(\dot{x}_{\min})$, $f_1^{-1}(\dot{x}_{\max})$, $f_2^{-1}(\dot{y}_{\min})$ and $f_2^{-1}(\dot{y}_{\max})$, we conclude that $P \cap \ell_a \neq \varnothing$ so that $S_a \neq \varnothing$.

   We show similarly the existence of Δ_b, Δ_c and Δ_d. In all the figures that come with this proof, the important line ℓ_a (as well as the lines ℓ_b, ℓ_c and ℓ_d corresponding to b, c and d respectively) is dashed.

2. We prove the optimality of the cut computed by Algorithm 6. Let

$$\widetilde{\Delta} = \frac{r_y}{2} - \Delta_0 \tag{7.3}$$

   and consider the following cases:

(2a) If $\Delta_1 > \widetilde{\Delta}$. The algorithm stops at line 13 and returns the line $\ell \equiv f_2(x,y) = \dot{y}_{\min} + \frac{r_y}{2}$. Consider Figure 7.8. Let $r \in P \cap [b', c']$ and $t \in P \cap [a', d']$. Obviously, the points r and t exist. Let $P/\ell = \langle P_r, P_t \rangle$ with $r \in P_r$ and $t \in P_t$.

   According to line 14 of the algorithm, let $Q_{\min} = P \cap [a', b']$ and $Q_{\max} = P \cap [c', d']$. We claim that $Q_{\min} \cap [a'', b''] = \varnothing$ and $Q_{\max} \cap [c'', d''] = \varnothing$. Since $\Delta_1 > \widetilde{\Delta}$, we

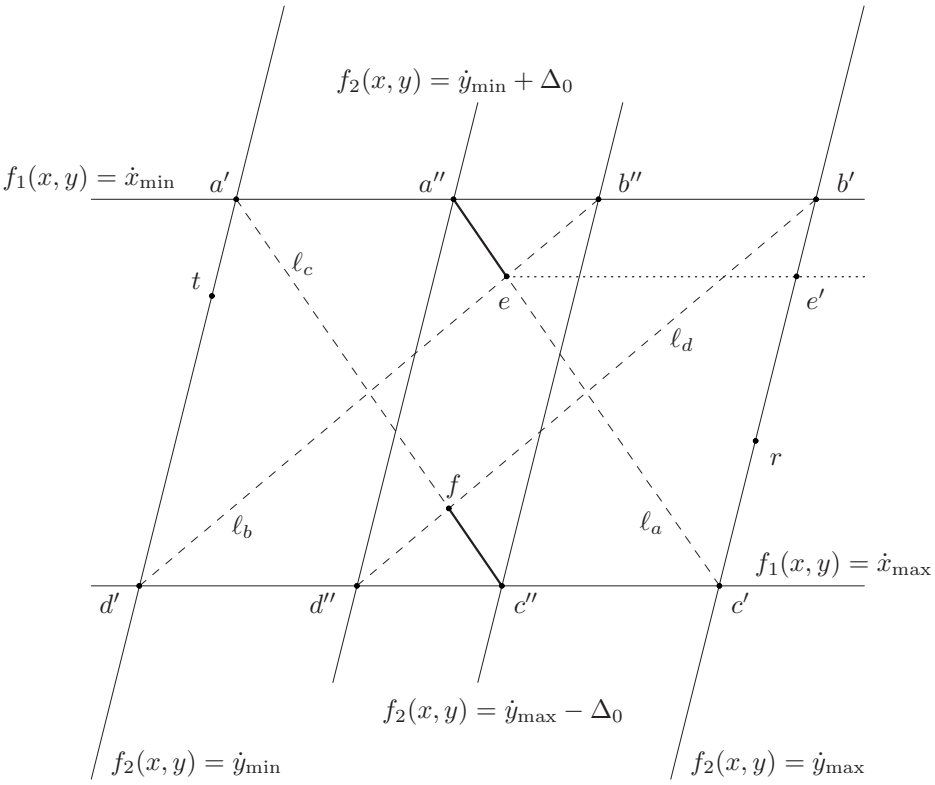


Figure 7.8: An example of the case $\Delta_1 > \widetilde{\Delta}$. The thick edges $[a'', e]$ and $[c'', f]$ are known to be outside P.

have $\Delta_a > \widetilde{\Delta}$ and $\Delta_c > \widetilde{\Delta}$. Let e (resp. f) be the intersection point of $[a'', c']$ and $[b'', d']$ (resp. $[a', c'']$ and $[b', d'']$). Note that the line returned by the algorithm is $\ell = \ell ine(e, f)$. It is easy to show that $e = a_{\widetilde{\Delta}} = b_{\widetilde{\Delta}}$ (and $f = c_{\widetilde{\Delta}} = d_{\widetilde{\Delta}}$) and therefore $P \cap [a'', e] = \varnothing$ and $P \cap [c'', f] = \varnothing$ (in thick lines on Figure 7.8). We apply Lemma 7.22 with $C = P \cap T$ where T is the triangle $a'b''d'$. Since $P \cap [a', d'] \neq \varnothing$, we have $P \cap [a'', b''] = \varnothing$ which entails our claim. Thus, either

- $Q_{\min} \subseteq [a', a'']$ and $Q_{\max} \subseteq [d', d'']$. See Figure 7.9(a). Then, we have:

$$\begin{array}{ll} \mathsf{sizeRange}_x^F(P_r) < r_x & \mathsf{sizeRange}_x^F(P_t) = r_x \\ \mathsf{sizeRange}_y^F(P_r) = \frac{r_y}{2} & \mathsf{sizeRange}_y^F(P_t) = \frac{r_y}{2} \end{array}$$

  Since $r_x > \frac{r_y}{2}$ (Relation 7.1), we have $\mathsf{sizeRange}^F(P/\ell) = r_x$. Now, assume (*ad absurdum*) that there is a line cut $\ell^\star$ such that $\mathsf{sizeRange}^F(P/\ell^\star) < \mathsf{sizeRange}^F(P/\ell)$. Let $P/\ell^\star = \langle P_1, P_2 \rangle$ with $r \in P_1$. From the fact that $\ell^\star$ is better than ℓ, we have $\forall z \in P_1 : f_2(r) - f_2(z) < r_x$ and by Equation 7.2, $f_2(z) > \dot{y}_{\min} + \Delta_0$. Therefore, we must have $Q_{\min} \subseteq P_2$ and $Q_{\max} \subseteq P_2$ so

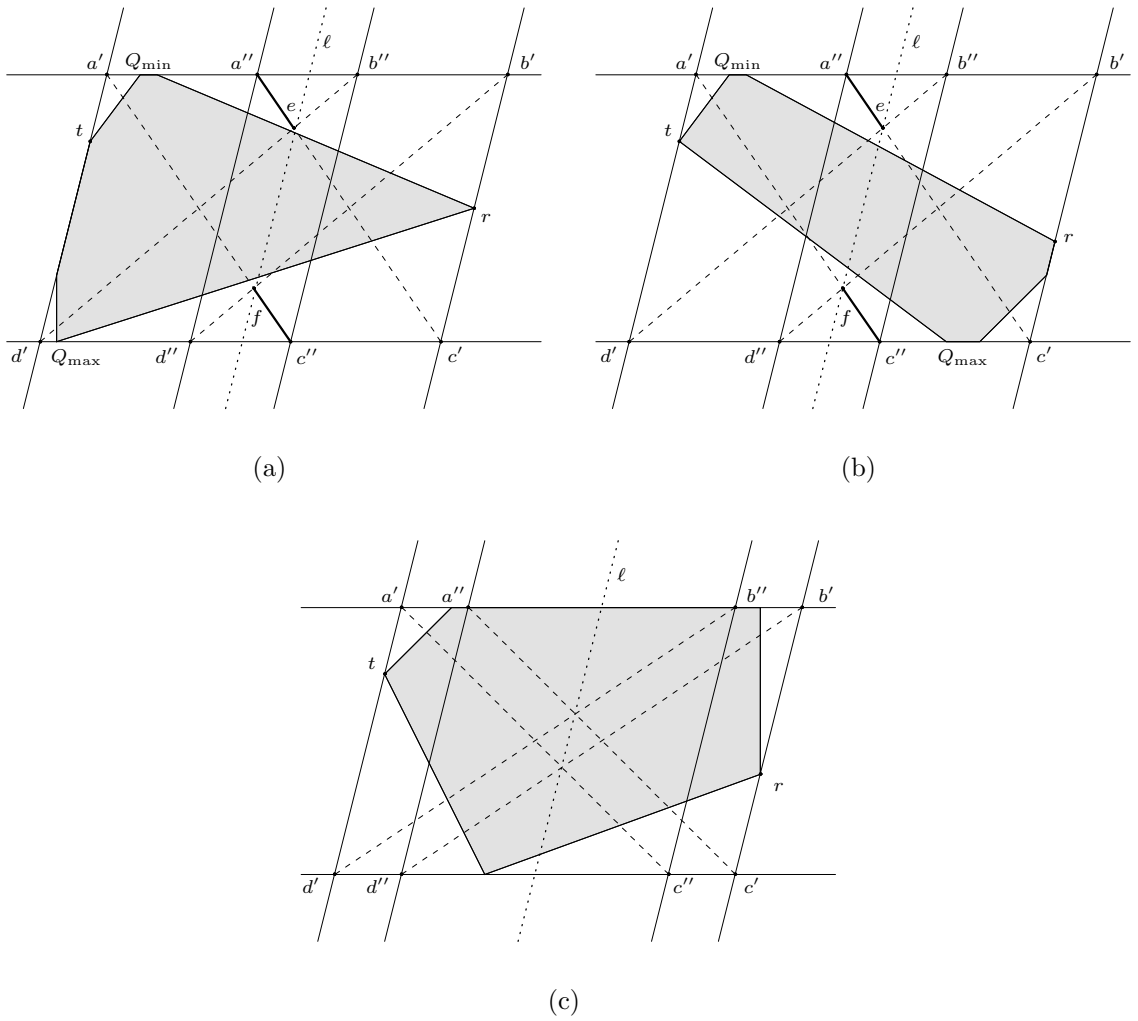


Figure 7.9: Examples.

that $\mathsf{sizeRange}^F(P_2) \geq r_x = \mathsf{sizeRange}^F(P/\ell)$ which contradicts the existence of $\ell^\star$. Thus ℓ is optimal.

- or $Q_{\min} \subseteq [a', a'']$ and $Q_{\max} \subseteq [c', c'']$. See Figure 7.9(b). From Lemma 7.22 with $C = P \cap T$ where T is the trapezoid $a'b'e'e$ (see Figure 7.8), since $P \cap [a', a''] \neq \varnothing$, we have $P \cap a''b'e'e = \varnothing$ so that $\forall z \in P_r : f_1(z) > \dot{x}_{\min} + \widetilde{\Delta}$. Similarly, $\forall z \in P_t : f_1(z) < \dot{x}_{\max} - \widetilde{\Delta}$. Thus we have:

$$\begin{array}{ll} \mathsf{sizeRange}^F_x(P_r) < r_x - \widetilde{\Delta} & \mathsf{sizeRange}^F_x(P_t) < r_x - \widetilde{\Delta} \\ \mathsf{sizeRange}^F_y(P_r) = \frac{r_y}{2} & \mathsf{sizeRange}^F_y(P_t) = \frac{r_y}{2} \end{array}$$

Clearly, $r_x - \widetilde{\Delta} = \frac{r_y}{2}$ (by Equations 7.2 and 7.3) so that $\mathsf{sizeRange}^F(P/\ell) =$

$\frac{r_y}{2} = \frac{\mathsf{sizeRange}^F(P)}{2}$. From Lemma 7.20, there exists no better cut.

- or $Q_{\min} \subseteq [b', b'']$. This case is treated similarly to the previous ones.

(2b) If $\Delta_1 = \widetilde{\Delta}$. A slight modification in the proof of the previous case allows to conclude: since either $\Delta_a = \widetilde{\Delta}$ or $\Delta_c = \widetilde{\Delta}$ (or both), we may have $e \in P$ or $f \in P$. Still, we have $Q_{\min} \cap [a'', b''[= \varnothing$ and $Q_{\max} \cap [c'', d''[= \varnothing$, and thus the rest of the proof is similar to the previous case.

(2c) If $\Delta_2 \geq \widetilde{\Delta}$. The proof is similar to the previous cases.

(2d) If $\Delta_1 < \widetilde{\Delta}$ and $\Delta_2 < \widetilde{\Delta}$. Assume without loss of generality that $\Delta_1 = \Delta_a$, $\Delta_2 = \Delta_b$ and $\Delta_1 > \Delta_2$. The algorithm continues after line 14, and we have $Q_{\min} = P \cap f_1^{-1}(\dot{x}_{\min})$ and $Q_{\max} = P \cap f_1^{-1}(\dot{x}_{\max})$. Let $r \in P \cap [b', c']$ and $t \in P \cap [a', d']$. Obviously, the points r and t exist. See Figure 7.7.

- If $Q_{\min} \cap [a', a''] \neq \varnothing$ and $Q_{\min} \cap [b', b''] \neq \varnothing$. See Figure 7.9(c). Then, the test of line 15 succeeds and the line $\ell \equiv f_2(x, y) = \dot{y}_{\min} + \frac{r_y}{2}$ is returned. Let $P/\ell = \langle P_r, P_t \rangle$ with $r \in P_r$ and $t \in P_t$. We have either $Q_{\max} \cap P_r \neq \varnothing$ or $Q_{\max} \cap P_t \neq \varnothing$. Therefore, since $r_x > \frac{r_y}{2}$ (Relation 7.1), we have $\mathsf{sizeRange}^F(P/\ell) = r_x$. Now, assume (*ad absurdum*) that there exists a line cut $\ell^\star$ such that $\mathsf{sizeRange}^F(P/\ell^\star) < \mathsf{sizeRange}^F(P/\ell)$. Let $P/\ell^\star = \langle P_1, P_2 \rangle$ with $r \in P_1$. From the fact that $\ell^\star$ is better than ℓ, we have $\forall z \in P_1 : f_2(r) - f_2(z) < r_x$ and by Equation 7.2, $f_2(z) > \dot{y}_{\min} + \Delta_0$. Then, $[a', a''] \subseteq P_2$, and we have $Q_{\min} \cap P_2 \neq \varnothing$. Similarly, we show that $Q_{\min} \cap P_1 \neq \varnothing$. Since we necessarily have either $Q_{\max} \cap P_1 \neq \varnothing$ or $Q_{\max} \cap P_2 \neq \varnothing$, it must be that $\mathsf{sizeRange}^F(P/\ell^\star) \geq r_x = \mathsf{sizeRange}^F(P/\ell)$, which contradicts the existence of $\ell^\star$. Thus ℓ is optimal.
- Else, if $Q_{\min} \cap [a', a''] \neq \varnothing$.

  If $Q_{\max} \cap [d', d''] \neq \varnothing$. This case is treated at lines 18, 19. The proof is similar to the previous cases.

  Otherwise, the algorithm returns $\ell = \ell ine(b_{\Delta_2}, d_{\Delta_2})$ (Line 20). Let $P/\ell = \langle P_r, P_t \rangle$ with $r \in P_r$ and $t \in P_t$. Define b''' as on Figure 7.10. Notice that $\Delta_2 \leq \Delta_b$ and $\Delta_2 \leq \Delta_d$ so that $P \cap [b'', b_{\Delta_2}[= \varnothing$ and $P \cap [d'', d_{\Delta_2}[= \varnothing$. From Lemma 7.22 with $C = P \cap T$ where T is the trapezoid $b'b'''b_{\Delta_2}c'$, we get that $\forall z \in P_t : f_2(z) \leq \dot{y}_{\max} - \Delta_0 - \Delta_2$. Similarly, $\forall z \in P_t : f_1(z) \leq \dot{x}_{\max} - \Delta_2$ and $\forall z \in P_r : f_2(z) \geq \dot{y}_{\min} + \Delta_0 + \Delta_2 \ \wedge \ f_1(z) \geq \dot{x}_{\min} + \Delta_2$. Therefore,

$$\begin{array}{ll} \mathsf{sizeRange}_x^F(P_r) \leq r_x - \Delta_2 & \mathsf{sizeRange}_x^F(P_t) \leq r_x - \Delta_2 \\ \mathsf{sizeRange}_y^F(P_r) \leq r_y - \Delta_0 - \Delta_2 & \mathsf{sizeRange}_y^F(P_t) \leq r_y - \Delta_0 - \Delta_2 \end{array}$$

  From Equation 7.2, $r_x - \Delta_2 = r_y - \Delta_0 - \Delta_2$, and thus $\mathsf{sizeRange}^F(P/\ell) \leq r_x - \Delta_2$. By a similar reasoning as before, we can show that any line cut $\ell^\star$

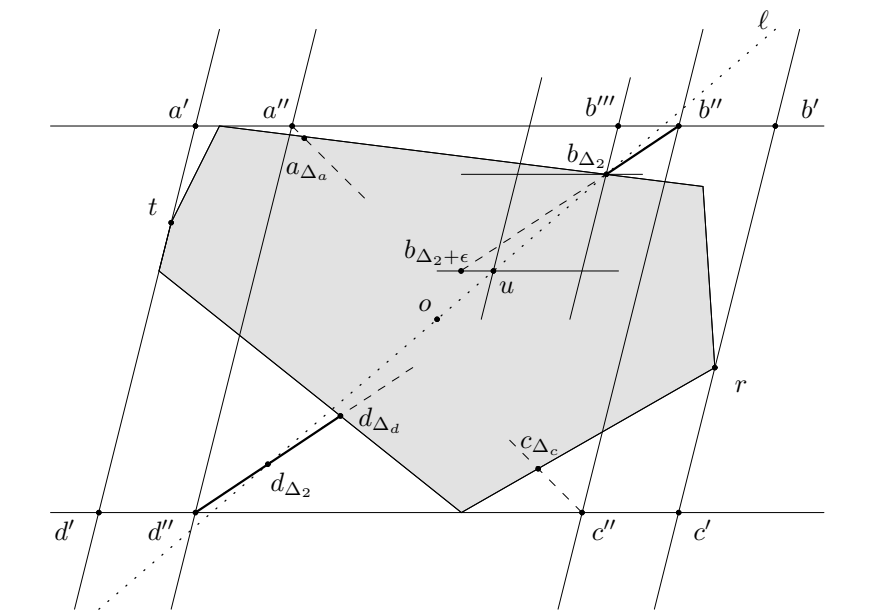


Figure 7.10: Constructions in the proof of Theorem 7.29.

such that $\mathsf{sizeRange}^F(P/\ell^\star) = r_x - \Delta_2 - \epsilon$ with $\epsilon > 0$ must separate Q_1^ϵ from Q_2^ϵ and R_1^ϵ from R_2^ϵ where:

$$\begin{aligned} Q_1^\epsilon &= P \cap f_1([\dot{x}_{\min}, \dot{x}_{\min} + \Delta_2 + \epsilon]) \\ Q_2^\epsilon &= P \cap f_2([\dot{x}_{\max} - \Delta_2 - \epsilon, \dot{x}_{\max}]) \\ R_1^\epsilon &= P \cap f_2([\dot{y}_{\min}, \dot{y}_{\min} + \Delta_0 + \Delta_2 + \epsilon]) \\ R_2^\epsilon &= P \cap f_2([\dot{y}_{\max} - \Delta_0 - \Delta_2 - \epsilon, \dot{y}_{\max}]) \end{aligned}$$

To do this, the line $\ell^\star$ should separate either $\{Q_1^\epsilon, R_1^\epsilon\}$ and $\{Q_2^\epsilon, R_2^\epsilon\}$ or $\{Q_1^\epsilon, R_2^\epsilon\}$ and $\{Q_2^\epsilon, R_1^\epsilon\}$. We prove that both cases are impossible so that $\ell^\star$ cannot exist.

We have $b_{\Delta_2} \in P$ and $o \in P$ by Lemma 7.23 (where o is the center of both parallelograms $a'b'c'd'$ and $a''b''c''d''$). Since P is convex, $[o, b_{\Delta_2}] \subseteq P$. We have:

$$\begin{aligned} f_1(b_{\Delta_2}) &= \dot{x}_{\min} + \Delta_2 & f_1(o) &= \dot{x}_{\min} + \tfrac{r_x}{2} \\ f_2(b_{\Delta_2}) &= \dot{y}_{\max} - \Delta_0 - \Delta_2 & f_2(o) &= \dot{y}_{\max} - \tfrac{r_y}{2} \end{aligned}$$

Since $\Delta_2 < \widetilde{\Delta}$, using Equation 7.2 and 7.3 we get $\Delta_2 < \frac{1}{2}(r_x - \Delta_0)$ and thus $\Delta_2 < \frac{r_x}{2}$. Assume now that $0 < \epsilon < \frac{r_x}{2} - \Delta_2$ and let $\alpha = \frac{\epsilon}{\frac{r_x}{2} - \Delta_2} < 1$. Consider the point $u = \alpha\, o + (1-\alpha) b_{\Delta_2}$. Clearly $u \in P$. Also, $f_1(u) = \alpha\, f_1(o) + (1-\alpha) f_1(b_{\Delta_2}) = \dot{x}_{\min} + \Delta_2 + \epsilon$ so that $u \in Q_1^\epsilon$ and

$$\begin{aligned} f_2(u) &= \alpha\, f_2(o) + (1-\alpha) f_2(b_{\Delta_2}) \\ &= \dot{y}_{\max} - \Delta_0 - \Delta_2 - \alpha\left(\tfrac{r_y}{2} - \Delta_0 - \Delta_2\right) \\ &= \dot{y}_{\max} - \Delta_0 - \Delta_2 - \alpha\left(\tfrac{r_x}{2} - \Delta_2 - \tfrac{\Delta_0}{2}\right) \\ &\geq \dot{y}_{\max} - \Delta_0 - \Delta_2 - \epsilon \end{aligned}$$

so that $u \in R_2^\epsilon$. Now, consider the parallelogram Π with sides parallel to the level sets of f_1 and f_2 and with diagonal $[u, b_{\Delta_2}]$. From Lemma 7.28, we have $\mathrm{int}(P \cap \Pi) \neq \varnothing$. Clearly, we have shown that $\Pi \subseteq Q_1^\epsilon$ and $\Pi \subseteq R_2^\epsilon$ so that $\mathrm{int}(Q_1^\epsilon \cap R_2^\epsilon) \neq \varnothing$. So, by Lemma 7.25 we have the inseparability of Q_1^ϵ and R_2^ϵ for $\epsilon < \frac{r_x}{2} - \Delta_2$. Since $Q_1^\epsilon \subseteq Q_1^\eta$ and $R_2^\epsilon \subseteq R_2^\eta$ for every $\epsilon \leq \eta$, we have inseparability of Q_1^ϵ and R_2^ϵ for every $\epsilon > 0$.

A similar proof shows that $\{Q_1^\epsilon, R_2^\epsilon\}$ and $\{Q_2^\epsilon, R_1^\epsilon\}$ are not separable because Q_1^ϵ and R_1^ϵ are not separable. This entails that $\ell^\star$ cannot exist.

- Else, if $Q_{\min} \cap [b', b''] \neq \varnothing$. The proof is similar to the previous case.
- Otherwise. The proof for the rest of the algorithm uses the same tricks as above.

■

7.5 Refinement-based Safety Verification Algorithm

In this section, we explain the methodology to obtain automatically successive refinements of an affine hybrid automaton H for which we want to check emptiness.

A first rectangular phase-portrait approximation $H_0 = \mathsf{rect}(H)$ is computed from the original automaton H. Then the automaton H_0 is symbolically analyzed, both forward and backward as described in Section 2.5. This gives the two sets $\mathsf{Reach}(\llbracket H_0 \rrbracket)$ and $\mathsf{Reach}^{-1}(\llbracket H_0 \rrbracket)$. If their intersection $\mathsf{Unsafe}(\llbracket H_0 \rrbracket)$ is empty, then so is the set $\mathsf{Unsafe}(\llbracket H \rrbracket)$ (by Lemma 7.3 and Theorem 7.4) and the emptiness of H is established. Otherwise, we refine the automaton H by splitting one of its unsafe locations and restart the procedure. In fact, we show that refining the rectangular approximation in a safe location is not helpful for checking emptiness. In other words, the relevant part of the state space to be refined is $\mathsf{Unsafe}(\llbracket H \rrbracket)$. Formally, the result follows from Theorem 7.30 where we use the notion of *pruning* of the state space. For an hybrid automaton $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ and a subset $L \subseteq \mathsf{Loc}$ of its locations, let $\mathsf{prune}(H, L)$ be the hybrid automaton $\langle \mathsf{Loc}', \mathsf{Lab}, \mathsf{Edg}', X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ where $\mathsf{Loc}' = \mathsf{Loc} \backslash L$, $\mathsf{Edg}' = \{(\ell, \sigma, \ell') \in \mathsf{Edg} \mid \ell, \ell' \in \mathsf{Loc}'\}$ and the other components are left unchanged.

Theorem 7.30 *For every hybrid automaton H, for every subset $L \subseteq \mathsf{SafeLoc}(H)$ of its safe locations, we have $\llbracket \mathsf{prune}(H, L) \rrbracket \approx_{unsafe} \llbracket H \rrbracket$, that is, $\llbracket \mathsf{prune}(H, L) \rrbracket$ and $\llbracket H \rrbracket$ are weakly bisimilar for the unsafe behaviours.*

Proof. Consider the relation $R = \{(q_1, q_2) \mid q_1 = q_2 \in \mathsf{Unsafe}([\![H]\!])\}$. Let us show that R is a weak simulation relation for $[\![\mathsf{prune}(H, L)]\!] \succeq_{unsafe} [\![H]\!]$. For all $(q_1, q_2) \in R$ and all $\sigma \in \mathsf{Lab} \setminus \{\tau\} \cup \mathbb{R}^{\geq 0}$, if $q_2 \overset{\sigma}{\twoheadrightarrow} q_2'$ in $[\![H]\!]$ and $q_2' = (\ell, v) \in \mathsf{Unsafe}([\![H]\!])$, then $\ell \notin L$ by definition of $\mathsf{SafeLoc}(H)$. Therefore, we have $q_1 \overset{\sigma}{\twoheadrightarrow} q_1'$ in $[\![\mathsf{prune}(H, L)]\!]$ for $q_1' = q_2'$. The other conditions of Definition 7.1 are established similarly, and the proof that R^{-1} is a weak simulation relation for $[\![H]\!] \succeq_{unsafe} [\![\mathsf{prune}(H, L)]\!]$ is trivial. ■

By Theorem 7.30 and Lemma 7.6, we conclude that splitting a location that is safe in the rectangular approximation has no relevance.

Corollary 7.31 *For every hybrid automaton H, for all locations $\ell \in \mathsf{SafeLoc}(\mathsf{rect}(H))$, and all hyperplanes π, $\mathsf{rect}(H)$ is empty iff $\mathsf{rect}(\mathsf{split}(H, \ell, \pi))$ is empty.*

The core of the refinement based verification procedure is given below. Its correctness is justified by Corollary 7.31. There is no guarantee of termination for two reasons. First, the reachability analyses of the rectangular approximations (that we perform for checking the condition of the *while* loop) may not terminate as the problem is undecidable. Second, the affine automaton H may be empty in the exact dynamics, but nonempty for any over-approximation, no matter how close of H it is. So, however the splitting can reduce the imprecision below any $\epsilon > 0$, the emptiness of H could not be established within a finite number of steps. However, we could stop when the size of the invariants run below a certain threshold and conclude that the system is at least not robustly correct for that threshold.

```
1 while Unsafe([[rect(H)]]) ≠ ∅ do
2 |   L ← SafeLoc(rect(H)) ;
3 |   H' ← prune(H, L) ;
4 |   Let ℓ be a location of H' and π be a hyperplane ;
5 |   H ← split(H', ℓ, π) ;
  |
```

The splitting is done in the location ℓ having the greatest imprecision (the largest value of sizeRange on its invariant) and the hyperplane π is determined using one of the algorithms presented in Section 7.4. This is a natural choice for ℓ since our goal is to finally reduce the overall imprecision of the rectangular approximation. This way, the approximation can be made arbitrarily close to the original system [HHW98] and so, if the system is robust against small perturbations as in [Frä99], our procedure eventually establishes its correctness, provided the reachability analysis of the rectangular automata terminates. This contrasts with the counter-example based refinement abstraction method (CEGAR) developed by Clarke *et al.* [CGJ+00] where the approximations are finite-state, but the refinement procedure is driven by the elimination of

spurious counter-examples (executions of the approximation which have no concrete counterpart) and therefore not guaranteed to terminate.

We have implemented a prototype tool to evaluate the refinement-based safety verification algorithm. Driven by practical experiments, we have brought some improvements to the theoretical algorithm. First, to reduce the number of iterations of the loop (and thus the number of analyses of rectangular approximations), we may perform more than one split at each iteration: we execute line 4 and line 5 repeatedly a fixed number of times (called *splits per iteration*). Second, the pruning can be improved as follows: for a location ℓ that is unsafe in $\mathsf{rect}(H)$, we can replace in H' the invariant of ℓ by any over-approximation of the unsafe states in ℓ, that is a polytope P such that $\forall(\ell, v) \in \mathsf{Unsafe}([\![\mathsf{rect}(H)]\!]) : v \in P$. In practice, we take P as the convex hull of the unsafe states in ℓ.

7.6 Case study

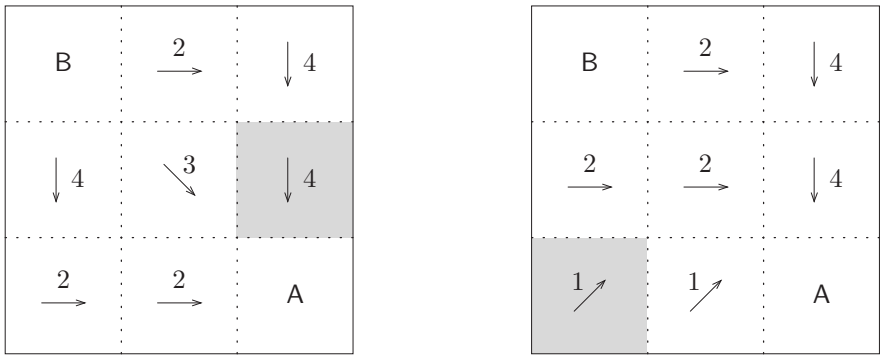
Our prototype tool is implemented in `C++` on top of the Parma Polyhedra Library (PPL) [BRZH02]. The PPL provides routines to handle convex polyhedra in exact arithmetic (using rational coefficients stored as pairs of integers with unbounded size). Linear programming with the simplex algorithm is one of the features.

For the analysis of rectangular automata, we make external calls to the model-checker PHAVer [Fre05], a recent tool for the verification of hybrid systems. This tool can handle affine dynamics hybrid automata, but the dynamics are first over-approximated by rectangular inclusions. The verification algorithm computes the reachable states with a procedure similar to HyTech that we have explained in Section 2.5, however drastically improved and based on PPL. The tool PHAVer provides an on-the-fly over-approximation of the affine dynamics, with refinements by location splitting with user-defined hyperplanes. Since we have an algorithm to automatically obtain the splitting hyperplanes, we do not use the refinement features of PHAVer. Also, we use the reverse automaton construction of Section 2.1 to implement backward analysis.

We applied our methodology to a benchmark, that we briefly present below. Our implementation is designed to specifically handle that benchmark (the example is hard-coded). However, the techniques that we have presented are general and the goal of the prototype is just to validate the approach in practice.

Navigation benchmark The Navigation benchmark is a set of thirty instances of the same problem, that was proposed in [FI04].

An object is moving on a $m \times n$ grid divided in $m \cdot n$ *cells*. The parameters m and n vary from 3 to 25 in the instances. The dynamics of the object are given by the



(a) NAV01-03 (b) NAV04

Figure 7.11: Navigation benchmark: the grid of desired velocities.

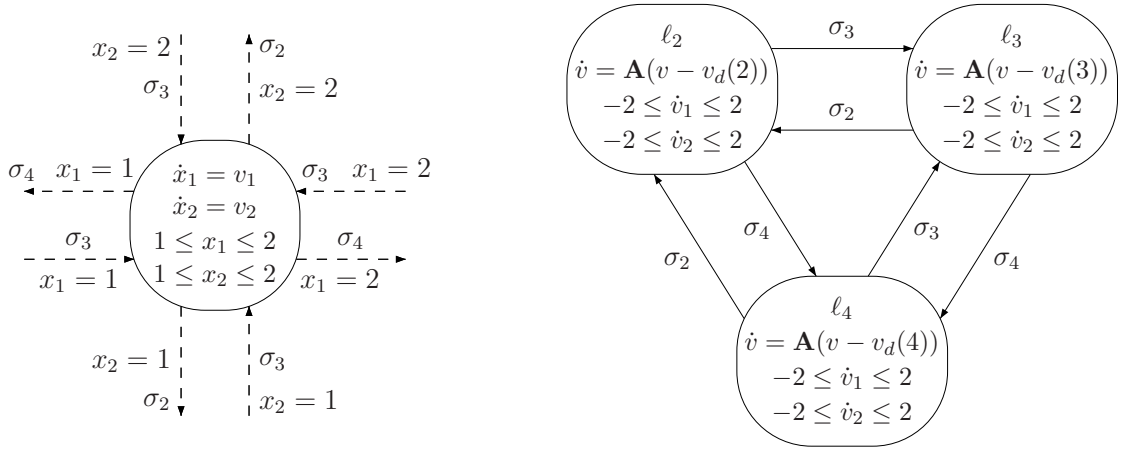
equations:

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= \mathbf{A}(v - v_d(i)) \end{aligned} \tag{7.4}$$

where $x = (x_1, x_2)^T$ records the position of the object on the grid, and $v = (v_1, v_2)^T$ records its velocity. The 2×2 matrix $\mathbf{A}$ is constant for each instance. The vector $v_d(i)$ models a *desired velocity* that is constant in each cell of the grid. Its value is determined by an integer $i \in \{0, \ldots, 7\}$ associated to each cell by a given map M as follows: $v_d(i) = (sin(i\pi/4), cos(i\pi/4))$. The grid with desired velocities for the instances NAV01-03 and NAV04 of the benchmark are shown in Figure 7.11. The eigenvalues of $\mathbf{A}$ have strictly negative real part, which guarantees that v eventually converges to v_d. The benchmark defines two particular cells, one labelled by A that has to be reached, and one labelled by B that is to be avoided. We are only interested in verifying the safety property that the B-cell is unreachable. The initial positions are shaded in Figure 7.11, the initial velocities (v_1, v_2) lie in the range:

NAV01: $[-0.3, 0.3] \times [-0.3, 0]$ NAV03: $[-0.4, 0.4] \times [-0.4, 0.4]$
NAV02: $[-0.3, 0.3] \times [-0.3, 0.3]$ NAV04: $[0.1, 0.5] \times [0.05, 0.25]$

The modeling of the Navigation benchmark with hybrid automata is depicted in Figure 7.12. We use the models of [Fre05]: an automaton Pos with $m \cdot n$ locations and variables x_1 and x_2, corresponding to the cells of the grid (Figure 7.12(a) shows the location for the central cell of NAV01-03), and an automaton Vel with a location for each desired velocity (Figure 7.12(b) shows the three modes that are present in NAV01-03). The velocities are *a priori* bounded to the interval $[-2, 2]$, to comply with the assumption that invariants are bounded. We check after the reachability analysis that the velocities are indeed such bounded. The synchronization labels σ_i, $i \in \{0, \ldots, 7\}$, give the index of the desired velocity according to the map M.



(a) Affine automaton Pos.

(b) Affine automaton Vel.

Figure 7.12: Hybrid automata for the Navigation benchmark.

Instance	Time	Memory	Splits/iteration	Nbr. iter.
NAV01	5s	42 MB	15	1
NAV02	7s	42 MB	15	1
NAV03	7s	42 MB	15	1
NAV04	60s	105 MB	15	4

Table 7.1: Navigation benchmark: Execution times and memory consumption on a Xeon 3GHz with 4GB RAM.

The Navigation benchmark has four variables x_1, x_2, v_1 and v_2, so we have to adapt the presentation of Section 7.4.2 that was in 2D. Observe that only two variables appear in the right-hand side of the differential equations 7.4, namely v_1 and v_2. So, the quality of the splitting is not influenced by the range of values of the position variables x_1 and x_2. Therefore, it will be sufficient to split in the plane v_1v_2 of the velocities (and thus in the automaton Vel), and we take $X = \{v_1, v_2\}$ for the input variable set of Algorithm 6. For the tuple of functions $F = \langle f_1, f_2 \rangle$, we may choose among the four functions that define the dynamics: v_1, v_2, $\mathbf{A}_{1*} \cdot (v - v_d)$ or $\mathbf{A}_{2*} \cdot (v - v_d(i))$ where $\mathbf{A}_{1*}$ (resp. $\mathbf{A}_{2*}$) is the first (resp. second) row of $\mathbf{A}$. In practice, we have observed that the first two functions give better results in that the fixed point computation of the reachable states converges much faster.

The execution time and memory consumption for the instances NAV01-04 are shown in Table 7.1. For instance NAV01-03, the safety of the system is established after one iteration of the refinement-based verification algorithm. The number of splits

per iteration is to be understood as the number of line cuts in each of the locations of the automaton Vel. So, for NAV01-03, the splitted automaton Vel has $48 = 3.(15 + 1)$ locations. For NAV04, we give the details of the computation in Table 7.2. For each iteration, there are two external calls to PHAVer, one for the forward analysis (F) and one for the backward analysis (B). For each call, we give the execution time, the memory consumption and the number of iterations for the fixed point in PHAVer. In Figure 7.13, we show the reachable states for each line of Table 7.2, except the first forward iteration, where the whole grid is reachable. The initial and bad states are darker (initial states darkest). For backward analysis, the set of initial states is the convex hull of the bad states that were reachable at the previous forward iteration. The pruning policy of the safe states reduces the size of the state space and allows to focus the computational effort to the parts where it is useful. Figure 7.13(f) shows that the bad states are unreachable after the 4th forward iteration.

The results reported in Figure 7.1 compare favourably with the times/memory reported in [Fre05]: 35s (63MB), 41s (54MB), 62s (90MB), and 225s (116MB) for NAV01-04 respectively. So, we were able to verify more efficiently than PHAVer itself the instances NAV01-04, while we used that tool as a black box for analysis of rectangular automata (thus with all heuristics disabled).

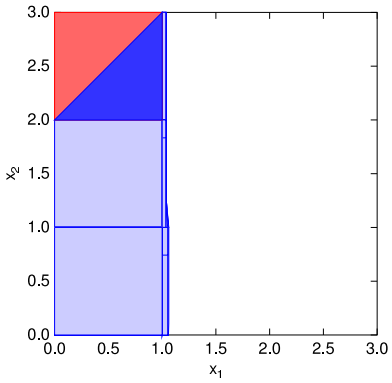
Iteration	F/B	Time	Memory	Nbr. iter. in PHAVer
1	F	5s	42 MB	14
1	B	5s	39 MB	8
2	F	8s	57 MB	10
2	B	10s	51 MB	15
3	F	7s	81 MB	9
3	B	4s	30 MB	7
4	F	21s	105 MB	11

Table 7.2: Navigation benchmark: computation details for NAV04.

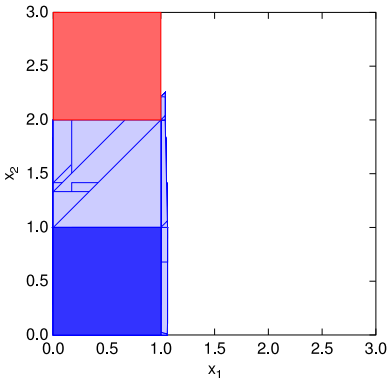
7.7 Conclusion and Future Works

Beyond Rectangular Approximations In the previous sections, the use of rectangular automata to approximate affine dynamics was motivated by the fact that (*i*) rectangles are simple shapes, the value of each variable can be chosen independently of the others (a rectangle is a Cartesian product of intervals), (*ii*) any level of precision can be obtained for the approximation, and (*iii*) the reachability analysis of rectangular automata is reasonably efficient.

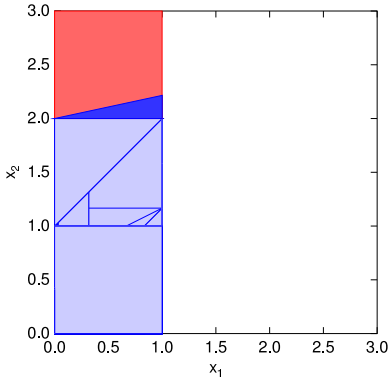
However, the more general class of linear hybrid automata is also analyzable in practice and it may happen that a linear dynamics approximation is sufficient to establish



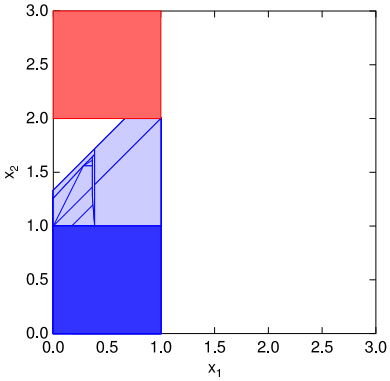
(a) 1st backward iteration.



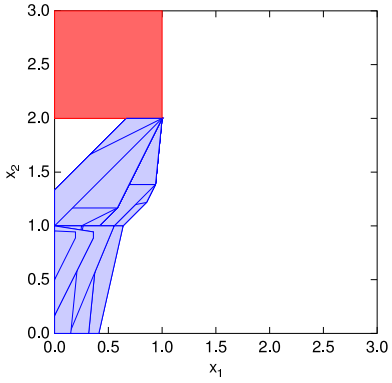
(b) 2nd forward iteration.



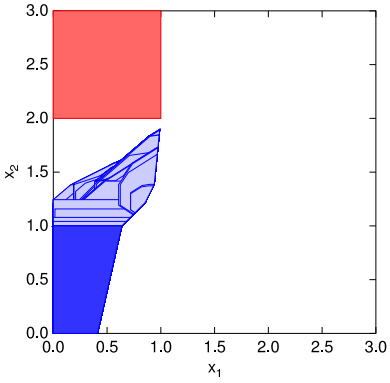
(c) 2nd backward iteration.



(d) 3rd forward iteration.



(e) 3rd backward iteration.



(f) 4th forward iteration.

Figure 7.13: Navigation benchmark: verification of NAV04 (see also Figure 7.11(b)).

emptiness while rectangular approximations fail, no matter the number of splits.

We illustrate this fact on an example. Consider the affine hybrid automaton H_1 of Figure 7.14. Let $P = [\![\mathsf{Inv}(\ell)]\!]$ be the invariant of the initial location. Let $\mathsf{Flow}_\ell : P \to \mathbb{R}^2 : (x, y) \to (x+1, x+y+1)$ be the affine function that gives the flow vector $(\dot{x}, \dot{y})$ in each point (x, y) of P. Since P is a polytope, the set $Q = \mathsf{Flow}(P)$ is also a polytope and is called the *dynamics polytope*. The set Q is shown on Figure 7.14.

This polytope corresponds to the dynamics of the linear hybrid automaton of Figure 7.15 that over-approximates H_1. We see that this approximation is already sufficient to prove that the location Bad is unreachable. Indeed, we have initially $x \leq y$ and the constraint $\dot{x} \leq \dot{y}$ ensures that $x \leq y$ is always true in the initial location. Therefore the guard $x > y$ is never enabled and Bad is unreachable.

On the other hand, if we cover the dynamics polytope with any finite number of rectangles, we cannot obtain that result. It is easy to show that there would necessarily be a (non-degenerated) rectangle that contains the dashed box of Figure 7.14 for ϵ sufficiently small. Therefore, from the initial point $x = y = 0$, there exist a dynamics in the rectangular approximation such that $\dot{x} > \dot{y}$ and therefore the guard $x > y$ is satisfied after any positive amount of time. Hence it is impossible to prove emptiness of H_1 with rectangular dynamics.

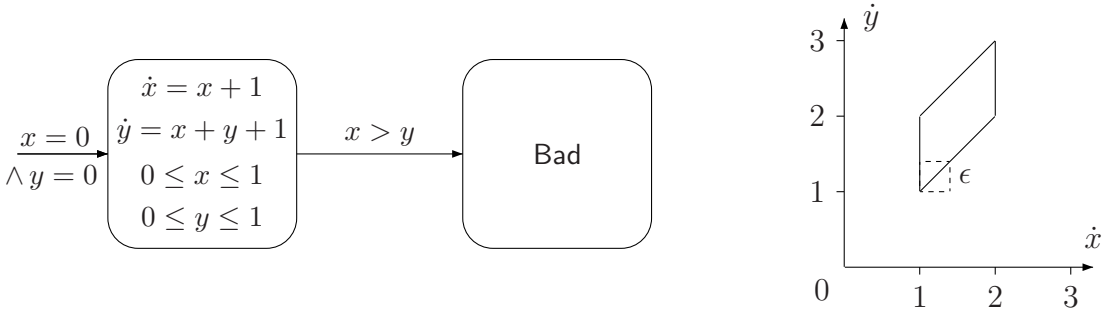


Figure 7.14: An affine hybrid automaton H_1 and its polytope of dynamics.

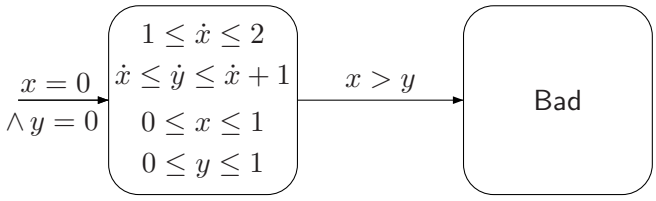


Figure 7.15: Approximation of H_1 by a linear hybrid automaton.

Refinement of linear approximations To automate the process of refinement with linear automata, we have to define a splitting policy. We would like to have an

algorithm for splitting in an optimal way, according to some criterion to be defined. So far, we do not know such an algorithm. Instead, we propose a heuristics based on the following observation.

We say that a set $C \subseteq \mathbb{R}^n$ is a *cone* if for all $x \in C$ and for all $\lambda \in \mathbb{R}^{\geq 0}$, $\lambda x \in C$. The *conic hull* of a set S is the smallest convex cone containing S. For two polyhedrons $P, Q \subseteq \mathbb{R}^n$, it is easy to see that:

$$P \nearrow Q = \{p + t.q \mid p \in P, q \in Q, t \in \mathbb{R}^{\geq 0}\} = \{p + t.\lambda q \mid p \in P, q \in Q, t, \lambda \in \mathbb{R}^{\geq 0}\}$$

and thus the set of time successors is not changed when the linear dynamics is replaced by its conic hull. More precisely, for a linear automaton H, let H' be the automaton obtained from H by replacing each predicate $\mathsf{Flow}(\ell)$ by a linear predicate φ_ℓ such that $[\![\varphi_\ell]\!]$ is the conic hull of $[\![\mathsf{Flow}(\ell)]\!]$. Then H and H' are time-abstract bisimilar, that is $\mathsf{Untime}([\![H]\!]) \approx \mathsf{Untime}([\![H']\!])$.

Therefore, conic hulls can be seen as a canonical representation of linear dynamics. The refinement policy should then be expressed in terms of those conic hulls. In the heuristics that we propose, we first split the cone of dynamics by a hyperplane yielding two pieces F_1 and F_2. Then, we cover the invariant by two regions P_1 and P_2 such that the dynamics polytope of P_1 (resp. P_2) is contained in F_1 (resp. F_2). The heuristics is to determine the splitting hyperplane. Observe that it is always preferable to choose a hyperplane passing by the origin (the point with all zero coordinates) to avoid overlapping of the two cone dynamics. Given a cone C, we split along the hyperplane $\pi \equiv (v - v')^T x = 0$ where v, v' are vectors of C such that $\|v\|_2 = \|v'\|_2$ and $angle(v, v') = \max\{angle(v_1, v_2) \mid v_1, v_2 \in C\}$. Thus we separate the vectors that are spanning the largest angle.

If C is the conic hull of a polytope Q, the maximal angle is realized by two vertices of Q. Therefore, computing v and v' is a matter of enumerating the pairs of vertices of Q. In practice, we may stop the refinement process when $\|v - v'\|_2$ run below some threshold. Finally, note that the splitting hyperplane is in general not definable by rational coefficients because the scaling of v and v' (such that $\|v\|_2 = \|v'\|_2$) may introduce irrational numbers.

Chapter 8

Conclusion

> Les hommes sont faits, nous dit-on,
> Pour vivre en bande comme les moutons,
> Moi j'vis seul et c'est pas demain
> Que je suivrai leur droit chemin.
>
> Georges Brassens, *La mauvaise herbe*.

8.1 Summary

We have considered two central formalisms for modelling real-time systems: timed and hybrid automata. We have briefly presented the existing (un)decidability results and algorithmic tools for the analysis of those models, with the focus on safety properties. We have observed that there is a price to pay for those results, among others:

- For timed automata, the classical semantics is idealized: the clocks have a perfect rate and guards are exact. This entails decidability of the emptiness problem. However, the classical semantics is not well suited for reasoning about real implementations.
- For hybrid automata, the dynamics is limited to rectangular or linear relations in the space of the derivatives. This entails computability of the one-step successors of a set of states. For practical applications however, the dynamics is often specified with more general differential equations.

In this thesis, we have studied the decidability of richer semantics for timed automata and more general dynamics for hybrid automata. We have also contributed to extend the algorithmic techniques for dealing with those semantics. We briefly summarize our main contributions for timed automata and hybrid automata:

- We have considered a first attempt to define a semantics that is decidable and suitable for implementations, where the use of equality in timing constraints is disallowed. Unfortunately, we have shown in Chapter 3 that even in that case, the parametric reachability problem for timed automata is undecidable. This result contrasts with the field of real-time logics where relaxing punctuality leads to decidability [AFH96]. This result has oriented our research to the AASAP semantics, a new semantics for timed automata that is implementable.

- For the verification of the AASAP semantics, we have a result of decidability for the emptiness problem, when the parameter δ is fixed to a rational constant. We have presented two algorithmic approaches to synthesize the value of δ.

- For the essential problem of deciding the existence of a value for the parameter δ in the AASAP semantics, we have made a detailed study of important properties of timed automata, and we have established that the problem is decidable. The result is related to the robustness of timed automata against perturbations in either the guards, the clock rate or both.

- We have proposed a refinement-based algorithm for the verification of safety properties for hybrid automata with affine dynamics. The algorithm approximates the dynamics by rectangular inclusions, which gives an automaton that can be analyzed with classical procedures. The approximation is conservative in that if the rectangular automaton satisfies a safety property, then so does the original automaton. Refined approximations are obtained automatically, according to an optimality criterion. For automata with two continuous variables, we compute the optimal refinement, and for automata in higher dimension, an arbitrarily precise approximation of the optimal refinement is computable. The method is further improved by running both a forward and backward exploration, and by pruning parts of the state space that are not relevant to the verification of the target property. Promising results have been obtained with an implementation prototype.

8.2 Future works

Several extensions of the work presented in this thesis are possible. We briefly summarize some directions for future works.

1. The programs that we have studied in this thesis belong to the category of *reactive* programs that have input/output interactions with an environment, and that are not intended to terminate. They differ from the *transformational* programs that are aimed at computing a result from a given input in a finite amount of time. For this kind of program, there exists a fairly standard *model of computation*:

Turing Machines. It is an abstract model of program that is not used for physical realization, but for theoretical reasonings about programs. Turing Machines are equivalent in expressive power to many other models of computation, and therefore they are widely accepted as the reference of what is *computable* by a digital device.

For reactive real-time programs, there is a lack of a similar abstract model of what is *implementable* on a digital device. There exist a lot of notations for specifying reactive systems, but none is sufficiently abstract to convince of its implementability in general. This is in contrast with classical models of computation. The presence of quantitative real-time constraints is the crucial feature. The solution is perhaps to be searched in the theory of timed languages, for identifying in the abstract a general class of *implementable languages*. We hope that the AASAP semantics and its motivations could help in the definition of such an abstract *model of reactive real-time computation*.

2. We have seen that many problems about parametric timed automata are undecidable. However, we have also shown that an important problem was decidable, namely when either guards are uniformly *enlarged* or when clocks *drift*. Other results of (un)decidability for particular classes of timed systems can probably be found, as slight changes in the definitions may turn an undecidable problem to a decidable one. For example, a recent result shows that the question whether there exists a sampling period such that a given timed automaton is empty is decidable when the automaton can idle [KP05]. Remind that without this assumption, the problem is undecidable [CHR02].

3. For most of the problems that are decidable for parametric timed automata, *efficient implementations* are still lacking. Future research should investigate the design of data structures and algorithms for the practical verification of implementable real-time systems. Techniques such as acceleration or abstract interpretation could be helpful in that context. It would also be useful to define other notions of implementability, and to find necessary or sufficient conditions of implementability that are easy to compute.

4. It would be interesting to continue investigating the analysis of hybrid automata with affine dynamics, and extend the techniques to *more complex dynamics*. In the theory of refinements, future developments include a better understanding of what makes the quality a refinement and how to construct them efficiently. The combination of abstraction-refinement with other techniques such as ellipsoids [KV00], barrier-certificates [PJ04], and theorem proving [RS05] should be explored. The question of the *scalability* of the methods for hybrid automata is also very important in practice.

Appendix A

Technical proofs

A.1 Proof of Theorem 4.7

Theorem 4.7 *Let $\mathcal{S}_1, \mathcal{S}_2$ be two input-enabled and composable* STTS *(structured by $(\Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau)$ and $(\Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau)$ respectively). Let $\rightarrow^1$, $\rightarrow^2$ and $\rightarrow$ be the transition relations of $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_1 \| \mathcal{S}_2$ respectively. If $(q_1, q_2) \in \mathsf{Reach}(\mathcal{S}_1 \| \mathcal{S}_2)$ and there is a discrete transition $(q_1, \sigma, q_1') \in \rightarrow^1$ with $\sigma \notin \Sigma^1_{\mathsf{in}}$ (respectively $(q_2, \sigma, q_2') \in \rightarrow^2$ with $\sigma \notin \Sigma^2_{\mathsf{in}}$) then there exists a state q_2' (resp. a state q_1') such that $((q_1, q_2), \sigma, (q_1', q_2')) \in \rightarrow$.*

Proof. We prove the first part of the theorem. The proof of the second part is completely similar. Let $\mathcal{S}_1$ be given by $\langle Q^1, Q^1_0, Q^1_f, \Sigma^1, \rightarrow^1 \rangle$ structured by $(\Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau)$ and $\mathcal{S}_2$ be given by $\langle Q^2, Q^2_0, Q^2_f, \Sigma^2, \rightarrow^2 \rangle$ structured by $(\Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau)$.

Let $(q_1, q_2) \in Q^1 \times Q^2$ and let $\sigma \in \Sigma^1_{\mathsf{out}} \cup \Sigma^1_\tau$ such that $(q_1, \sigma, q_1') \in \rightarrow^1$. First, if $\sigma \in \Sigma^1_\tau$ then by Definition 4.4 we have $((q_1, q_2), \sigma, (q_1', q_2')) \in \rightarrow$ for $q_2' = q_2$. Second, if $\sigma \in \Sigma^1_{\mathsf{out}}$ then we consider two cases:

1. If $\sigma \notin \Sigma^2_{\mathsf{in}} \cup \Sigma^2_{\mathsf{out}} \cup \Sigma^2_\tau$, then by Definition 4.4 we have $((q_1, q_2), \sigma, (q_1', q_2')) \in \rightarrow$ for $q_2' = q_2$.

2. If $\sigma \in \Sigma^2_{\mathsf{in}} \cup \Sigma^2_{\mathsf{out}} \cup \Sigma^2_\tau$, then since $\mathcal{S}_1$ and $\mathcal{S}_2$ are composable, we have $\sigma \in \Sigma^2_{\mathsf{in}}$. As $\mathcal{S}_2$ is input enabled, by Definition 4.2 there exists $q_2' \in Q^2$ such that $(q_2, \sigma, q_2') \in \rightarrow^2$. Hence, by Definition 4.4 we have$((q_1, q_2), \sigma, (q_1', q_2')) \in \rightarrow$.

■

A.2 A Note on Analytic Functions

A real function $f : \mathbb{R} \to \mathbb{R}$ is said *non-trivial* if it is not identically 0. A function $f : \mathbb{R} \to \mathbb{R}$ is *infinitely derivable* iff its nth derivative $f^{(n)}$ exists for every $n \geq 1$. We denote by $C^\infty(\mathbb{R})$ the set of infinitely derivable functions. Given a function $f \in C^\infty(\mathbb{R})$ and a point $x_0 \in \mathbb{R}$, the *Taylor series* $T^f_{x_0}$ of f at x_0 is given by

$$T^f_{x_0}(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

with the convention that $f^{(0)} = f$.

Definition A.1 [Analytic function] A function f is called *analytic* if $f \in C^\infty(\mathbb{R})$ and for every $x_0 \in \mathbb{R}^n$, there exists a neighborhood of x_0 (a set containing an open set containing x_0) in which $f(x)$ is equal to its Taylor series at x_0. □

Theorem A.2 *The set $Z_f = \{x \mid f(x) = 0\}$ of the zeroes of a non-trivial analytic function is* isolated, *that is for all $x_0 \in Z_f$ there exists a neighborhood V_{x_0} of x_0 such that $V_{x_0} \cap Z_f = \{x_0\}$.*

Proof. Let $x_0 \in Z_f$. Assume (*ad absurdum*) that the intersection of every neighborhood of x_0 with Z_f contains another point than x_0. Let V_{x_0} be a neighborhood of x_0 such that $f(x) = T^f_{x_0}(x)$ in V_{x_0}.

We have $f(x_0) = 0$ and we show by induction that $f^{(k)}(x_0) = 0$ for all $k \geq 1$. Assume that $f^{(k)}(x_0) = 0$ for $k < n$. Then for all $x \in V_{x_0}$:

$$f(x) = \sum_{k=n}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k = (x-x_0)^n \underbrace{\left[\frac{f^n(x_0)}{n!} + g(x)\right]}_{R(x)} \text{ where } g(x) \to 0 \text{ for } x \to x_0$$

Since $g(x)$ can be made arbitrarily small in a neighborhood of x_0, if $f^n(x_0) \neq 0$ then $R(x) \neq 0$ in some neighborhood of x_0. Therefore f has only one zero in such a neighborhood (namely x_0). This contradiction yields $f^n(x_0) = 0$ and so $T^f_{x_0}(x)$ (and thus also f) is identically 0 on V_{x_0}.

Now, let U be the set of points x such that f vanishes identically on some neighborhood of x. The existence of V_{x_0} shows that U is not empty and clearly U is open. Let u be a boundary point of U. Since $f \in C^\infty(\mathbb{R})$, we have $f^{(k)}(u) = 0$ for all $k \in \mathbb{N}$. Thus the Taylor series of f at u is identically 0 and so is f in a neighborhood of u. Then $u \in U$ and U is also closed so that $U = \mathbb{R}$ and f vanishes identically on $\mathbb{R}$, a contradiction. ■

Corollary A.3 *A non-trivial analytic function has a finite number of zeroes over any bounded interval* $[a, b] \subseteq \mathbb{R}$.

Bibliography

[AAB00] Aurore Annichini, Eugene Asarin, and Ahmed Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV 2000)*, pages 419–434, 2000.

[AB01] Eugene Asarin and Ahmed Bouajjani. Perturbed turing machines and hybrid systems. In *Proc. 16th Annual Symposium on Logic in Computer Science (LICS)*, pages 269–278. IEEE Computer Society Press, 2001.

[ACD+92] Rajeev Alur, Costas Courcoubetis, David L. Dill, Nicolas Halbwachs, and Howard Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proc. 13th Real-time Systems Symposium*, pages 157–166. IEEE Computer Society Press, 1992.

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

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

[ADMB00] Eugène Asarin, Thao Dang, Oded Maler, and Olivier Bournez. Approximate reachability analysis of piecewise-linear dynamical systems. In *Proc. HSCC 00: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1790, pages 20–31. Springer-Verlag, 2000.

[AETP99] Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron Peled. Parametric temporal logic for "model measuring". In *Proc. of ICALP 99: Automata, Languages and Programming*, Lecture Notes in Computer Science 1644, pages 159–168. Springer-Verlag, 1999.

[AFH96] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43:116–146, 1996.

[AFM+02] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times: A tool for modelling and implementation of embedded

systems. In J.-P. Katoen and P. Stevens, editors, *Proc. Of The 8 Th International Conference On Tools And Algorithms For The Construction And Analysis Of Systems*, number 2280 in Lecture Notes In Computer Science, pages 460–464. Springer–Verlag, 2002.

[AFP+03] Tobias Amnell, Elena Fersman, Paul Pettersson, Hongyan Sun, and Wang Yi. Code synthesis for timed automata. *Nordic Journal of Computing(NJC)*, 9(4), 2003.

[AH92] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 74–106. Springer, 1992.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *ACM Symposium on Theory of Computing*, pages 592–601, 1993.

[AIK+03] Rajeev Alur, Franjo Ivancic, Jesung Kim, Insup Lee, and Oleg Sokolsky. Generating embedded software from hierarchical hybrid models. In *LCTES 03: Languages, Compilers, and Tools for Embedded Systems*, pages 171–182. ACM, 2003.

[AL94] Martín Abadi and Leslie Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, September 1994.

[AMPS98] Eugène Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proc. System Structure and Control*. Elsevier, 1998.

[AS85] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[AT04] Manindra Agrawal and P. S. Thiagarajan. Lazy rectangular hybrid automata. In *Proc. of HSCC 04: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 2993, pages 1–15. Springer-Verlag, 2004.

[AT05] Karine Altisen and Stavros Tripakis. Implementation of timed automata: an issue of semantics or modeling? In *Proc. of FORMATS 2005: Formal Modelling and Analysis of Timed Systems*, Lecture Notes in Computer Science 3829, pages 273–288. Springer-Verlag, 2005.

[ATM05] Rajeev Alur, Salvatore La Torre, and Parthasarathy Madhusudan. Perturbed timed automata. In *Proc. of HSCC 05: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 3414, pages 70–85. Springer-Verlag, 2005.

[BDR03] Véronique Bruyère, Emmanuel Dall'Olio, and Jean-François Raskin. Durations, parametric model-checking in timed automata with presburger arithmetic. In *Proc. of STACS 03: Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 2607. Springer-Verlag, 2003.

[Ber00] Gérard Berry. *The Foundations of* ESTEREL. MIT Press, 2000.

[Bey01] Dirk Beyer. Improvements in BDD-based reachability analysis of timed automata. In *Proc. of FM 2001: Formal Methods"*, Lecture Notes in Computer Science 2021, pages 318–343. Springer-Verlag, 2001.

[BG92] Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[BLN03] Dirk Beyer, Claus Lewerentz, and Andreas Noack. RABBIT: A tool for BDD-based verification of real-time systems. In *Proc. of CAV 2003: Computer Aided Verification*, Lecture Notes in Computer Science 2725, pages 122–125. Springer-Verlag, 2003.

[Bou01] Patricia Bouyer. Updatable timed automata, an algorithmic approach. Technical Report LSV-01-12, ENS Cachan, Cachan, France, 2001.

[BR03] Véronique Bruyère and Jean-François Raskin. Real-time model-checking: Parameters everywhere. In *Proc. of FSTTCS 03: Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science 2914, pages 100–111. Springer-Verlag, 2003.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[BRZH02] R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In M. V. Hermenegildo and G. Puebla, editors, *Static Analysis: Proc. of the 9th International Symposium*, Lecture Notes in Computer Science 2477, pages 213–229, Madrid, Spain, 2002. Springer-Verlag, Berlin.

[BS91] Janusz A. Brzozowski and Carl-Johan H. Seger. Advances in asynchronous circuit theory. Part II: Bounded inertial delay models, MOS circuits, design techniques. *Bulletin of the EATCS*, 43:199–263, 1991.

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proc. of CAV 2000: Computer Aided Verification*, Lecture Notes in Computer Science 1855, pages 154–169. Springer-Verlag, 2000.

[CGP99] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[Che68] N.V. Chernikova. Algorithm for discovering the set of all solutions of a linear programming problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.

[CHR91] Zhou Chaochen, C. A. R. Hoare, and Anders P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

[CHR02] Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In *Proc. of HSCC 02: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 2289, pages 134–148. Springer-Verlag, 2002.

[CHS93] Zhou Chaochen, Michael R. Hansen, and Peter Sestoft. Decidability and undecidability results for duration calculus. In *In Proc. of STACS 93: Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 665, pages 58–68. Springer-Verlag, 1993.

[CY91] Costas Courcoubetis and Mihalis Yannakakis. Minimum and maximum delay problems in real-time systems. In *Proc. 3rd Int. Workshop Computer Aided Verification (CAV'91)*, Lecture Notes in Computer Science 575, pages 399–409. Springer-Verlag, 1991.

[DDMR04] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robustness and implementability of timed automata. In *Proc. of FORMATS-FTRTFT '04*, Lecture Notes in Computer Science 3253, pages 118–133. Springer-Verlag, 2004.

[DDR04] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics: From timed models to timed implementations. In *Proc. of HSCC 04: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 2993, pages 296–310. Springer-Verlag, 2004.

[DDR05a] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics: From timed models to timed implementations. *Formal Aspects of Computing*, 17(3):319–341, 2005.

[DDR05b] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Systematic implementation of real-time models. In *Proc. of FM 2005: Formal Methods*, Lecture Notes in Computer Science 3582, pages 139–156. Springer-Verlag, 2005.

[DE73] George B. Dantzig and B. Curtis Eaves. Fourier-motzkin elimination and its dual. *Journal of Combinatorial Theory*, 14:288–297, 1973.

[DHR05] Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Automatic rectangular refinement of affine hybrid systems. In *Proc. of FORMATS 2005: Formal Modelling and Analysis of Timed Systems*, Lecture Notes in Computer Science 3829, pages 144–161. Springer-Verlag, 2005.

[Die99] Henning Dierks. *Specification and Verification of Polling Real-Time Systems*. PhD thesis, University of Oldenburg, July 1999.

[Die01] Henning Dierks. PLC-automata: a new class of implementable real-time automata. *Theoretical Computer Science*, 253(1):61–93, 2001.

[Dil89] David Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. 1st Int. Workshop Automatic Verification Methods for Finite State Systems (CAV'89)*, Lecture Notes in Computer Science 407, pages 197–212. Springer-Verlag, 1989.

[DOTY95] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool KRONOS. In *Proc. of Hybrid Systems III: Verification and Control*, Lecture Notes in Computer Science 1066, pages 208–219. Springer-Verlag, 1995.

[ET99] E. Allen Emerson and Richard J. Trefler. Parametric quantitative temporal reasoning. In *Proc. 14th Annual Symposium on Logic in Computer Science (LICS)*, pages 336–343. IEEE Computer Society Press, 1999.

[FI04] Ansgar Fehnker and Franjo Ivancic. Benchmarks for hybrid systems verification. In *Proc. of HSCC 04: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 2993, pages 326–341. Springer-Verlag, 2004.

[FR75] Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.

[Frä99] Martin Fränzle. Analysis of hybrid systems: An ounce of realism can save an infinity of states. In *CSL*, Lecture Notes in Computer Science 1683, pages 126–140. Springer-Verlag, 1999.

[Fre05] Goran Frehse. PHAVER: Algorithmic verification of hybrid systems past hytech. In *Proc. of HSCC 05: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 3414, pages 258–273. Springer-Verlag, 2005.

[GHJ97] Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan. Robust timed automata. In *HART 97: Hybrid and Real-Time Systems*, Lecture Notes in Computer Science 1201, pages 331–345. Springer-Verlag, 1997.

[Hal93] Nicolas Halbwachs. *Synchronous programming of reactive systems*. Kluwer Academic Pub., 1993.

[HCRP91] Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous dataflow programming language LUSTRE. *Proc. of the IEEE*, 79(9):1305–1320, September 1991.

[Hen92] Thomas A. Henzinger. Sooner is safer than later. *Information Processing Letters*, 43:135–141, 1992.

[Hen00] Thomas A. Henzinger. The theory of hybrid automata. In M.K. Inan and R.P. Kurshan, editors, *Verification of Digital and Hybrid Systems*, NATO ASI Series F: Computer and Systems Sciences 170, pages 265–292. Springer-Verlag, 2000.

[HHK01] Thomas A. Henzinger, Benjamin Horowitz, and Christoph M. Kirsch. GIOTTO: A time-triggered language for embedded programming. In *EMSOFT 01: Embedded Software*, Lecture Notes in Computer Science 2211, pages 166–184. Springer-Verlag, 2001.

[HHW95] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A user guide to HYTECH. In *TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1019, pages 41–71. Springer-Verlag, 1995.

[HHW98] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.

[HK02] Thomas A. Henzinger and Christoph M. Kirsch. The Embedded Machine: Predictable, portable real-time code. In *Proc. of the International Conference on Programming Language Design and Implementation*, pages 315–326. ACM Press, 2002.

[HKLC04] Yerang Hur, Jesung Kim, Insup Lee, and Jin-Young Choi. Sound code generation from communicating hybrid models. In *Proc. of HSCC 04: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 2993, pages 432–447. Springer-Verlag, 2004.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.

[HL02] Martijn Hendriks and Kim G. Larsen. Exact acceleration of real-time model checking. In *Proc. of TPTS'02*, volume 65 of *ENTCS*. Elsevier Science, 2002.

[HMP05] Thomas A. Henzinger, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying similarities between timed systems. In *Proc. of FORMATS 2005: Formal Modelling and Analysis of Timed Systems*, Lecture Notes in Computer Science 3829, pages 226–241. Springer-Verlag, 2005.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

[HPR94] Nicolas Halbwachs, Yann-Eric Proy, and Pascal Raymond. Verification of linear hybrid systems by means of convex approximations. In *International Static Analysis Symposium, SAS'94*, Namur (Belgium), September 1994.

[HR00] Thomas A. Henzinger and Jean-François Raskin. Robust undecidability of timed and hybrid systems. In *Proc. of HSCC 00: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1790, pages 145–159. Springer-Verlag, 2000.

[HRSV01] Thomas Hune, Judi Romijn, Marielle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. In *Proc. 7th Int. Conf. Tools and Algorithms for Construction and Analysis of Systems (TACAS'01)*, pages 189–203, 2001.

[HVG04] Jinfeng Huang, Jeroen Voeten, and Marc Geilen. Real-time property preservation in concurrent real-time systems. In *Proc. of RTCSA 04: Real-time and Embedded Computing Systems and Applications*, Lecture Notes in Computer Science. Springer-Verlag, 2004.

[IKL+00] T. Iversen, K. Kristoffersen, K. G. Larsen, M. Laursen, R. Madsen, S. Mortensen, P. Petterson, and C. Thomasen. Model-checking real-time control programs – verifying LEGO mindstorms systems using UPPAAL. In *Proc. of ECRTS '00*, pages 147–155. IEEE, 2000.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104. Plenum Press, New York, 1972.

[KMTY04] Pavel Krcál, Leonid Mokrushin, P. S. Thiagarajan, and Wang Yi. Timed vs. time-triggered automata. In *CONCUR 2004: 15th Int. Conf. Concurrency Theory*, Lecture Notes in Computer Science 3170, pages 340–354. Springer-Verlag, 2004.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[KP05] Pavel Krcál and Radek Pelánek. On sampled semantics of timed systems. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science 3821, pages 310–321. Springer-Verlag, 2005.

[KV00] Alexander B. Kurzhanski and Pravin Varaiya. Ellipsoidal techniques for reachability analysis. In *Proc. of HSCC 00: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1790, pages 202–214. Springer-Verlag, 2000.

[Lab02] Jean J. Labrosse. *MicroC/OS-II. The Real-Time Kernel*. CMP Books, 2002.

[Leg04] François Legros. Vérification de contrôleurs ELASTIC: mise en pratique. Master's thesis, Université Libre de Bruxelles, 2004.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT: Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[LPY01] Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. Symbolic reachability computation for families of linear vector fields. *Journal of Symbolic Computation*, 32(3):231–253, 2001.

[LT87] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *6th ACM Symp. on Principles of Distributed Computing*, pages 137–151, 1987.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science 92. Springer-Verlag, 1980.

[Mil00] Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In *Proc. of HSCC 00: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1790, pages 296–309. Springer-Verlag, 2000.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, London, 1967.

[Par81] David Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. of 5th GI conference on Theoretical Computer Science*, Lecture Notes in Computer Science 104, pages 167–183. Springer-Verlag, 1981.

[PJ04] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Proc. of HSCC 04: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 2993, pages 477–492. Springer-Verlag, 2004.

[Pur98] Anuj Puri. Dynamical properties of timed automata. In *FTRTFT '98*, Lecture Notes in Computer Science 1486, pages 210–227. Springer-Verlag, 1998.

[RS05] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *Proc. of HSCC 05: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 3414, pages 573–589. Springer-Verlag, 2005.

[San99] Marco A. A. Sanvido. A computer system for model helicopter flight control, part 3: The software core. Technical Report 317, Inst. Computer Systems, ETH Zürich, 1999.

[Sta02] Thomas Stauner. Discrete-time refinement of hybrid automata. In *Proc. of HSCC 02: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 2289, pages 407–420. Springer-Verlag, 2002.

[Wan95] Farn Wang. Timing behavior analysis for real-time systems. In *Proc. 10th Annual Symposium on Logic in Computer Science (LICS)*, pages 112–122. IEEE Computer Society Press, 1995.

[WH97] Farn Wang and Pao-Ann Hsiung. Parametric analysis of computer systems. In *Proc. of AMAST 97: Algebraic Methodology and Software Technology*, Lecture Notes in Computer Science 1349, pages 539–553. Springer-Verlag, 1997.

[Yov96] Sergio Yovine. Model checking timed automata. In *European Educational Forum: School on Embedded Systems*, pages 114–152, 1996.

Index

Symbols

A

B

C

D

U

V

W

Z