From Qualitative to Quantitative Analysis of Timed Systems

Mémoire d'habilitation à diriger des recherches

Patricia Bouyer

Defended on January 12, 2009

The committee was composed of:

Prof. Rajeev Alur	(rapporteur)
Prof. Ahmed Bouajjani	
Prof. Joost-Pieter Katoen	(rapporteur)
Prof. Kim G. Larsen	
Prof. Anca Muscholl	(rapporteur)
Prof. Antoine Petit	
Prof. Jean-François Raskin	

This thesis is mostly based on joint works with:

Christel Baier	Technische Universität Dresden	Germany
Gerd Behrmann	Aalborg Universitet	Denmark
Nathalie Bertrand	IRISA/INRIA Rennes	France
Thomas Brihaye	Université de Mons-Hainaut	Belgium
Ed Brinksma	Twente Universiteit	The Netherlands
Véronique Bruyère	Université de Mons-Hainaut	Belgium
Fabrice Chevalier	LSV/CNRS & ENS Cachan	France
Emmanuel Fleury	LaBRI/Université Bordeaux 1	France
Marcus Größer	Technische Universität Dresden	Germany
Kim G. Larsen	Aalborg Universitet	Denmark
Nicolas Markey	LSV/CNRS & ENS Cachan	France
Joël Ouaknine	Oxford University	England
Radek Pelànek	Masarykova Univerzita, Brno	Czech Republic
Jean-François Raskin	Université Libre de Bruxelles	Belgium
Jacob I. Rasmussen	Aalborg Universitet	Denmark
James Worrell	Oxford University	England

Contents

Chapter 1

Introduction

This habilitation thesis reports on a selection of my contributions since my PhD thesis in 2002.

The growing importance of embedded systems

Embedded systems are a modern technology which has an important impact on our everyday life. There are indeed more and more computing systems, and our comfort more and more relies on them (as an illustration in thirty years, we have gone from 'no phone at home' to 'everyone with a mobile phone'). One of the characteristics shared by these systems is that they have to meet numerous quantitative constraints, like resource constraints (power consumption, memory usage, costs, bandwidth, *etc.*), timing constraints (response time, propagation delays, *etc.*), and constraints on the environment in which they operate (signal sensors, interactions with a (possibly continuous) environment, *etc.*). Another important characteristic of embedded systems is that they have to be powerful, performant, and reliable. Thus, their conception and verification pose a great challenge!

The model-checking approach to verification

As part of the effort that has been made for the development of reliable embedded systems, several verification approaches have been developed, among which the so-called *model-checking approach*, which I more specifically work on. Model-checking is a model-based approach to verification, and has been distinguished by the Turing award in 2007,[1] awarded to Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis.

[1] `http://www.acm.org/press-room/news-releases/turing-award-07/`

Given a system $\mathcal{S}$ and a property P to be checked on $\mathcal{S}$, the model-checking approach consists in constructing a mathematical model $\mathcal{M}_{\mathcal{S}}$ for the system and a mathematical model φ_P for the property, for which we will be able to *automatically* check that $\mathcal{M}_{\mathcal{S}}$ satisfies φ_P. If the models are accurate enough with respect to $\mathcal{S}$ and P, we will deduce that the system $\mathcal{S}$ satisfies the property P.

Models for representing systems can be automata, and extensions thereof, but also process algebra, Petri nets, *etc.*, and properties can either be modelled in the very same formalisms, or as formulas in some (*eg.* temporal) logic. For instance, one can write formulas like

$$\mathbf{G}\,(\mathsf{car.crash} \rightarrow \mathbf{F}\,\mathsf{airbag.inflate}) \tag{1.1}$$

to express that the airbag inflates, each time the car crashes.

A difficulty of this approach is that accurate (classes of) models need to be developed, that should be readable (for a human being to be able to write and understand the model), and that should be *efficiently* and automatically analysable. Of course, given a precise system to be analysed, the model which is chosen needs be a trade-off between expressiveness, conciseness, and analysability.

The challenges that model-checking techniques have to face are, among others: (*i*) the growing complexity of computing systems, (*ii*) their growing size yielding the well-known state-explosion problem, (*iii*) the requirement for correct systems, and (*iv*) a race towards high-performance systems. We give below some solutions the model-checking approach suggests to challenges posed by the development of embedded systems.

embedded systems challenges	**model-checking solutions**
complex computing systems	development of expressive (and concise) models
growing-size systems	development of abstractions to fight the state-explosion problem
critical aspects, correctness, safety	development of (efficient) model-checking algorithms, qualitative analysis (answer: 'no' or 'yes')
performance aspects	quantitative analysis (answer: a value)

In this thesis, we will (partly) address most of these aspects, in the framework of timed automata.

The timed automaton model

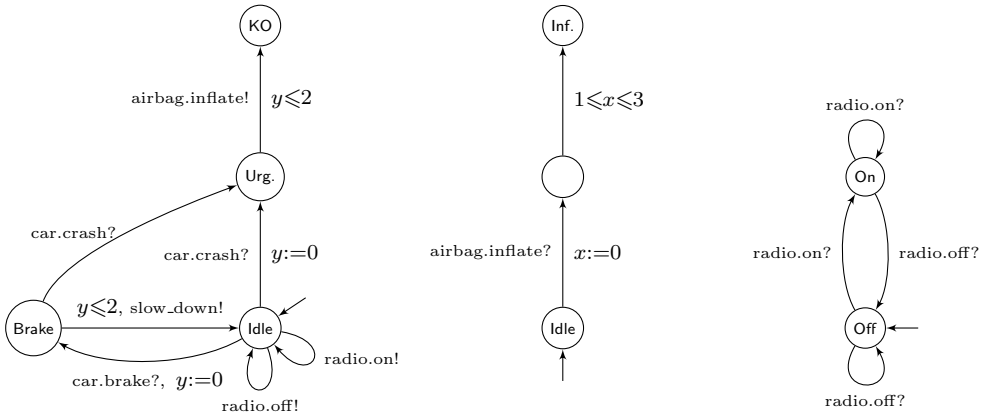
In this thesis, we will be interested in timed systems, *i.e.* in systems that have to meet quantitative constraints on delays between events, and not only constraints on the order of these events. For instance, in those systems, we may be more interested in the following property:

$$\mathbf{G}\,(\mathsf{car.crash} \rightarrow \mathbf{F}_{\leqslant 5\,\mathrm{ms}}\,\mathsf{airbag.inflate}) \tag{1.2}$$

which says that the airbag should inflate **no later than** $5\,\mathbf{ms}$ after the car crashes, than in the untimed property (1.1).

Even though several models taking time into account had already been proposed in the past, for instance time(d) Petri nets [Mer74, Ram74], in the early nineties, a great step has been done towards the development of verification techniques for timed systems, with the development of *timed automata* by Rajeev Alur and David Dill [AD90, AD94].

A timed automaton is a finite automaton that can manipulate variables. These variables have a very specific behaviour, and increase synchronously with time, they can be compared with a constant, or reset to 0. We give a (simplified) model for a car, as a network of timed automata.



Controller Airbag Radio

The timed automaton for the airbag reads as follows: the airbag is idle until it receives a signal saying that it should inflate, and within $[1, 3]$ after this signal is received, the airbag gets inflated. We use a binary synchronisation, an event $a!$ synchronises with an event $a?$. Note that the above system is not closed (some events, like 'slow_down?', 'car_brake!' are missing).

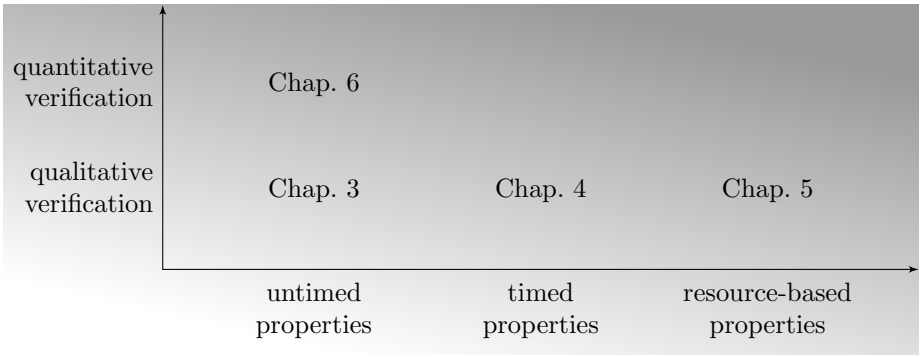
One of the fundamental properties of the timed automaton model is that, though there are infinitely many possible configurations in the system, many verification problems can be solved (*eg.* reachability and safety properties,

untimed ω-regular properties, branching-time timed temporal properties, *etc.*), because a finite-state abstraction, called the *region automaton*, can be used to check those properties. An extensive literature has been written on that model since the original paper [AD90], and it is worth noticing that Rajeev Alur and David Dill have received in 2008 the CAV award for this seminal article, which is among the most quoted article in computer-aided verification.[2] Also, developments have gone up to the implementation of tools like Uppaal [3] [BDL+06], and to their application to industrial case studies, for instance [HSLL97, BBP04, JRLD07].

Content of the thesis

The contributions I will describe follow a natural evolution in my research interests from the verification of simple (untimed) properties in timed systems to the verification of quantitative properties in those systems, not only involving constraints over delays, but also over resources. Also it follows another evolution from qualitative to quantitative verification: we start with questions like "does the system satisfy a property?", which requires a 'yes/no' answer, and end up with questions like "how likely will a system satisfy the property?", to which an answer should be a value. The latter question is very close to questions asked in performance evaluation.

A rough picture of the two evolutions is given below, and each technical chapter of this thesis is classified.



Chapter 2 shortly reports the basic definitions, that will be needed in the rest of the thesis.

[2] See `http://www.princeton.edu/cav2008/cav_award_announce.html`.

[3] See `http://www.uppaal.com/`

In chapter 3, we focus on the most basic properties one wants to verify, namely reachability (or basic safety) properties. These properties ask whether some location of the timed automaton can be reached (or on the contrary avoided). We present two generic approaches to verify those properties, and discuss their correctness and termination when applied to timed automata. Then, we present an abstraction operator that yields intriguing[4] but interesting correctness problems. We end the chapter with two improved abstraction operators, that have led to an impressive speed-up of the verification process, and are now implemented as standard options in the verification tool Uppaal.

These works have been started and partly done while visiting Aalborg Universitet (Denmark) during my post-doc. This stay has been a great opportunity to get interested in more practical aspects of the verification of timed systems, with a focus on data structures, algorithms and abstractions.

This chapter is based on the papers [Bou03, Bou04, BBFL03, BBLP04, BBLP05], and works on the improved abstractions are joint works with Gerd Behrmann, Emmanuel Fleury, Kim G. Larsen, and Radek Pelànek.

In chapter 4, we consider more involved properties of timed systems, which can express constraints over delays. We consider linear-time timed temporal logics as a formalism for expressing properties of systems. These logics extend the classical LTL [Pnu77] logic with timing constraints, and the property (1.2) is such a property. In that framework, with Fabrice Chevalier and Nicolas Markey, we started in 2005 working on a 15-year-old conjecture by Rajeev Alur and Thomas A. Henzinger [AH90], that there is an expressiveness gap between MTL [5] and TPTL,[6] two natural timed extensions of LTL. We have proven the conjecture, but we have seen that the tentative witness proposed by [AH90] needed to be refined. This piece of works, published as [BCM05, BCM08], is one of the first expressiveness results obtained in the framework of timed systems, and has inspired some other research, among which [DP06, DRP07].

In 2005, I have been quite fascinated by the paper by Joël Ouaknine and James Worrell at LICS'05 [OW05], who have proven that the already-mentioned logic MTL is decidable[7] (under some restricted semantics, though), contrary to what was actually believed in the community. The techniques

[4] The classical abstraction operator that was commonly used is actually not correct for the whole class of timed automata **with** diagonal constraints.

[5] MTL stands for "Metric Temporal Logic" and has been defined in [Koy90].

[6] TPTL stands for "Timed Propositional Temporal Logic" and has been defined in [AH89].

[7] It is however non-primitive recursive!

used, based on well-quasi-orders were also really interesting,[8] and that is when I really got interested in these logics for model-checking purposes. That is also the period when I decided to spend a sabbatical year at Oxford University (England),[9] which I have done in 2007. In this chapter, I explain (in a rather simple way) why it is difficult to use general linear-time timed logics in the verification process. I then present several restrictions that have been made to get improved decidability (and complexity) results, and terminate with a description of a fragment of MTL that is rather expressive and for which the model-checking problem can be solved rather efficiently [BMOW07, BMOW08]. These are joint works with Nicolas Markey, Joël Ouaknine, and James Worrell.

In chapter 5, timed automata are extended with 'observer variables', that evolve in a more complex way than clocks do, but cannot be used to restrict the possible executions of the system. These additional variables are called cost variables as they can be used to measure the cost (in terms of energy, or price, *etc.*) of an execution. They increase linearly with time in every location (with a rate depending on the location), and discretely when a transition is fired. This model has been defined in 2001 by Rajeev Alur *et al.* [ALP01] and Kim G. Larsen *et al.* [BFH$^+$01], and various optimisation and model-checking questions have been addressed since then on that model. For instance, one can ask what is the best way to execute a system in such a way that the cost per time unit is minimal. I have started working on that model while visiting Aalborg Universitet (Denmark) with Ed Brinksma and Kim G. Larsen, precisely on the previously mentioned optimisation problem, and I found it really challenging and exciting.

In this chapter, I will first present the model of weighted timed automata (this is how we will call this extension of timed automata with cost variables), and give the first decidability results that have been proven for basic optimisation problems. Then we extend to the two-player framework, which is a usual framework for modelling systems embedded in an environment [Thom02], where one of the players tries to minimize the cost, and the other player tries to maximize the cost. This problem will happen to be much more complex than the single-player framework that was implicitly assumed before. Indeed, in general the two-player problem is undecidable (we will explain the idea of the proof, as we find it rather instructive), and

[8] It is fair saying that such techniques had already been developed in the context of networks of similar single-clock timed automata [AJ98], and also in the context of timed Petri nets [AN01] by Parosh Aziz Abdulla, Bengt Jonsson, and Aletta Nylén.

[9] Where Joël Ouaknine and James Worrell hold a position.

only the restriction to single-clock automata yields decidability. Then, we investigate a natural model-checking problem, where properties no more express constraints on delays (as in chapter 4), but express constraints on the cost variables. We extend classical temporal logics with cost constraints and can write properties like

$$\mathbf{G}\,(\mathsf{failure} \rightarrow (\mathsf{out\_of\_order}\,\mathbf{U}_{\mathsf{cost}\leqslant 56\,€}\,\mathsf{repair}))$$

which says that each time there is a failure, the machine will be put into the out_of_order mode until it is repaired, and the reparation will cost no more than 56 €. Unfortunately, as in the two-player framework, the results are negative: the model-checking of both natural extensions of CTL and LTL with cost constraints is undecidable. However, restricting the number of clocks to one yields, as in the two-player framework, decidability of the model-checking problem, with a reasonable complexity (at least not higher than the corresponding problem with timing constraints instead of cost constraints).

This chapter reports, among others, works published as [BBL04, BBL08, BCFL04, BCFL05, BBM06, BM07, BBBR07, BLM07, BLM08], and which are joint works with Thomas Brihaye, Ed Brinksma, Véronique Bruyère, Franck Cassez, Emmanuel Fleury, Kim G. Larsen, Nicolas Markey, and Jean-François Raskin. Furthermore, we also briefly report ongoing works.

In chapter 6, I will present the most recent developments I have been involved in. This work has long been something I had in mind, but my knowledge in probabilistic systems was not strong enough to do it on my own. In 2006, together with Thomas Brihaye, who was holding a post-doctoral position in Cachan at that time, we decided to start working on the subject, and to get help from people who were specialists of probabilistic systems. That is why we visited Christel Baier's group in Dresden in December 2006.

The aim is to add probabilities to timed automata, so that *unlikely* (sequences of) events will happen with probability 0. The model of probabilistic timed automata developed by the PRISM team [KNSS02, KNP04] is not sufficient, because the delays are still chosen non-deterministically, and the idea is then to make all delay choices probabilistic instead. This yields a purely probabilistic semantics to timed automata, that somehow extends the continuous-time Markov chain model [Fel68]. In this framework, we first investigate the natural almost-sure model-checking problem, which asks whether a property is satisfied with probability one in a timed automaton. Due to the rather complex structure of the probabilistic space generated by a timed automaton, it was not easy to get decidability of the problem, and we have then defined an 'equivalent' topological space where

largeness[10] (of ω-regular sets of runs) coincides with sets of probability one, in the restricted framework of single-clock timed automata though. Using that characterisation we have designed an algorithm that decides the almost-sure model-checking problem for ω-regular properties. This algorithm is based on the construction of a finite Markov chain that is a correct abstraction for the almost-sure model-checking problem, but only in the single-clock automata framework. It is interesting to notice that this limitation to a single clock is unfortunately not only a restriction that appears in the proof, but that we have designed a two-clock counter-example to that algorithm. These works are joint works with Christel Baier, Nathalie Bertrand, Thomas Brihaye and Marcus Größer, and have been published as [BBB+07, BBB+08a]

Then, we have gone to the more complex quantitative model-checking problem, where the aim is to compute the probability (or approximations thereof) of a given ω-regular property in a timed automaton. The previous abstraction is no more correct in that framework, and we have designed another finite Markov chain abstraction that solves the quantitative model-checking problem for a restricted class of timed automata. This work will appear as [BBBM08] and is joint work with Nathalie Bertrand, Thomas Brihaye and Nicolas Markey.

This subject is rather new, I believe it is challenging and interesting. It leaves a wide range of open problems, notably the mix of non-determinism and probabilities, as in Markov decision processes. In the end of the chapter, I report some preliminary results that we are currently working on.

In chapter 7, I give some conclusions and perspectives, even though every chapter already gives perspectives for the specific subject developed in the chapter.

In the following, the symbol ☞ indicates one of my results.

Other recent contributions not included in this thesis

I will briefly review some of my other recent works, and refer to

`http://www.lsv.ens-cachan.fr/~bouyer/mes-publis.php`

for a complete list of my publications.

[10] In the topological sense. We recall that a set in a topological space is *meager* if it is a denumerable union of nowhere-dense sets, and a set is *large* if its complement is meager. Note that this is a stronger notion than density, because a dense set can be meager! This is for instance the case of $\mathbb{Q}$ in $\mathbb{R}$ (for the classical topology).

Time(d) Petri nets. With Serge Haddad[11] and my (now former) PhD student Pierre-Alain Reynier, we have investigated some timed and distributed systems. We have in particular studied several models that add timing constraints to Petri nets, the so-called time/timed Petri nets. We have compared their expressiveness with that of networks of timed automata, and have obtained interesting results for the timed Petri net model, where Zeno behaviours distinguish between timed automata and bounded timed Petri nets. All these works correspond to the articles [BHR06a, BHR06b, BHR08].

Using ideas taken from our study of timed Petri nets, we have proposed an algorithm that applies a partial-order reduction to networks of timed automata. This is a challenging issue, because timing constraints put dependency on events, that could appear as independent at first sight. This work has been published as [BHR06c].

Robustness and implementability issues. This issue shall be a bit discussed in chapter 6. What we have seen so far is that timed automata can be seen as a model for representing embedded systems. They can also be used to model programs. However, this model then appears as too idealized, because a real implementation platform will not assume the same hypotheses as the 'mathematical' model does. For instance, clocks in a timed automata are infinitely precise, whereas they are digital in a real processor, actions and communications are instantaneous in a timed automaton, whereas they are not in a real computing system, *etc.*, hence the necessity of integrating these aspects in the verification process. This can be done either by integrating the implementation platform in the model, as proposed in [AT05], or by over-approximating the behaviour of the real system by another timed automaton, as suggested by [DDR04]. In the latter, a model of robustness is proposed, that over-approximates the behaviour of the system implemented using a simple processor. The robustness notion coincides with the one proposed few years earlier by Anuj Puri [Pur98]. This notion of robustness has been further investigated, and an algorithm for verifying basic safety properties (under the robust semantics) has been proposed in [DDMR04, DDMR08], algorithm that we have extended to LTL properties in [BMR06] and even later to a subclass of timed properties in [BMR08]. Those are joint works with Nicolas Markey and Pierre-Alain Reynier.

O-minimal hybrid systems. I would like to mention a last subject, namely the model of o-minimal hybrid systems. We know that general hy-

[11] Who was at LAMSADE, Université Paris-Dauphine (France) at that time, and is now at LSV/CNRS & ENS Cachan (France).

brid systems are mostly undecidable [Hen96, HKPV98], and harsh restrictions need to be made to yield decidability. O-minimal hybrid systems have been first proposed in [LPS00] as an interesting class of systems. These systems have very rich continuous dynamics, but limited discrete steps (at each discrete step, all variables have to be reset, independently from their initial values). This allows to decouple the continuous and discrete components of the hybrid system, and to deduce properties of the global system from those of the continuous parts of the system.

With Thomas Brihaye and my (now former) PhD student Fabrice Chevalier, we have studied the two-player game problem in o-minimal hybrid systems, and have shown that it was decidable, proving that the suffix encoding proposed in [Bri07] was actually a correct abstraction for solving the problem. This has been published as [BBC06b]. Recently, a side result of a joint work with Marcin Jurdziński, Ranko Lazić and Michał Rutkowski (University of Warwick, England), and Thomas Brihaye [BBJ+08] is that the o-minimality hypothesis made in [BBC06b] is not required to get the decidability, and only the strong-reset assumption is sufficient.

Also, following the development of weighted timed automata (*cf.* chapter 5), we have extended o-minimal hybrid systems with cost (observer) variables, and have proven that most problems that are undecidable for weighted timed automata become decidable for that model, see [BBC07].

Chapter 2

Timed systems: things one should know before starting

2.1 The timed automaton model

We assume the framework of dense time, and refer to [Alu91, chapter 2] for a justification of that framework.

2.1.1 Preliminaries

We consider as time domain the set $\mathbb{R}_{\geqslant 0}$ of non-negative reals. Let Σ be a finite alphabet. A *timed word* over Σ is a finite or infinite sequence of pairs $(a_0, \tau_0)(a_1, \tau_1)(a_2, \tau_2)\cdots$ such that for every $i \in \mathbb{N}$, $a_i \in \Sigma$, and $(\tau_i)_{i\in\mathbb{N}}$ is a non-decreasing sequence in $\mathbb{R}_{\geqslant 0}$. Furthermore we may call the τ_i's the *time stamps* of the timed word.

We let X be a finite set of variables, called *clocks*. A *(clock) valuation* over X is a mapping $v : X \to \mathbb{R}_{\geqslant 0}$ that assigns to each clock a time value. The set of all valuations over X is denoted $\mathbb{R}_{\geqslant 0}^X$. Let $t \in \mathbb{R}_{\geqslant 0}$, the valuation $v + t$ is defined by $(v + t)(x) = v(x) + t$ for every $x \in X$. For $Y \subseteq X$, we denote by $[Y \leftarrow 0]v$ the valuation assigning 0 (respectively $v(x)$) for any $x \in Y$ (respectively $x \in X \setminus Y$). We write $\mathbf{0}_X$ for the valuation which assigns 0 to every clock $x \in X$.

The set of *(general) clock constraints* over X, denoted $\mathcal{C}(X)$, is defined by the grammar:

$$\mathcal{C}(X) \ni g \ ::= \ x \sim c \ \mid \ x - y \sim c \ \mid \ g \wedge g$$

where $x, y \in X$ are clocks, $c \in \mathbb{Q}$, and $\sim \in \{<, \leqslant, =, \geqslant, >\}$. Let $k \in \mathbb{Q}_{\geqslant 0}$, a clock constraint g is *k-bounded* if any constant c appearing in g lies within the interval $[-k; +k]$.

Clock constraints are evaluated over clock valuations, and the satisfaction relation, denoted $v \models g$, is defined inductively as follows:

$$\begin{cases} v \models (x \sim c) & \text{if} \quad v(x) \sim c \\ v \models (x - y \sim c) & \text{if} \quad v(x) - v(y) \sim c \\ v \models g_1 \wedge g_2 & \text{if} \quad v \models g_1 \text{ and } v \models g_2 \end{cases}$$

Also, we write $[\![g]\!] = \{v \in \mathbb{R}^X_{\geqslant 0} \mid v \models g\}$ the polyhedron defined by g. We denote by $\mathcal{C}_{df}(X)$ the subset of $\mathcal{C}(X)$ that do not make use of *diagonal constraints*, *i.e.* of constraints of the form $x - y \sim c$.

2.1.2 Timed automata

The model of timed automata has been defined initially by Alur and Dill in the early nineties [AD90, AD94]. In this thesis, we will consider various questions on timed automata, that may require several features that were not initially in the model (like invariants, labelling by atomic propositions, *etc.*). For convenience, we thus define a general model of timed automata including all these aspects, and might forget some of the useless components in the following chapters.

Definition 2.1 *A* timed automaton *is a tuple*

$$\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L})$$

where AP *is a finite set of atomic propositions,* X *is a finite set of clocks,* L *is a finite set of locations,* $\ell_0 \in L$ *is the initial location,* $\mathsf{Goal} \subseteq L$ *is a set of goal (or final) locations,* $E \subseteq L \times \mathcal{C}(X) \times 2^X \times L$ *is a finite set of edges (or transitions),* $\mathsf{Inv} : L \to \mathcal{C}_{df}(X)$ *assigns an invariant to every location, and* $\mathcal{L} : L \to 2^{\mathsf{AP}}$ *labels every location with atomic propositions.*

A timed automaton in which all clock constraints are diagonal-free, is said diagonal-free*.*

***Remark* 2.2** In the above definition, we omit actions labelling edges, contrary to classical definitions, because we will not use them in the following (but will use the name of the edge instead, for instance to define the transition system given by a timed automaton). However, they are useful for modelling purposes, see for instance the simplified car model on page 7. ⌟

The semantics of a timed automaton $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L})$ is given as a timed labelled transition system $\mathcal{T}_{\mathcal{A}} = (S, s_0, \to, \mathcal{L})$ where $S = L \times \mathbb{R}^X_{\geqslant 0}$ is the set of configurations (or states)[1] of $\mathcal{A}$, $s_0 = (\ell_0, \mathbf{0}_X)$ is the

[1] Later, we write $\mathsf{loc}(s) = \ell$ and $\mathsf{val}(s) = v$ whenever $s = (\ell, v)$.

initial configuration, $\mathcal{L}$ extends the labelling function of $\mathcal{A}$ to configurations, writing $\mathcal{L}(s) \stackrel{\text{def}}{=} \mathcal{L}(\mathsf{loc}(s))$, and $\to$ contains two types of moves:

- *delay moves:* $(\ell, v) \xrightarrow{t} (\ell, v+t)$ [2] if $t \in \mathbb{R}_{\geqslant 0}$, and for all $0 \leqslant t' \leqslant t$, $v + t' \models \mathsf{Inv}(\ell)$;
- *discrete moves:* $(\ell, v) \xrightarrow{e} (\ell', v')$ if there exists an edge $e = (\ell, g, Y, \ell')$ in E such that $v \models g \wedge \mathsf{Inv}(\ell)$, $v' = [Y \leftarrow 0]v$, and $v' \models \mathsf{Inv}(\ell')$.

A *run* ϱ in $\mathcal{A}$ is a finite or infinite sequence of moves in the transition system $\mathcal{T}_{\mathcal{A}}$, with a strict alternation of delay moves (though possibly 0-delay moves) and discrete moves. In the following, we may write a run $\varrho = s \xrightarrow{t_1} s'_1 \xrightarrow{e_1} s_1 \xrightarrow{t_2} s'_2 \xrightarrow{e_2} s_2 \ldots$ more compactly as $\varrho = s \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \ldots$. A transition of the form $s \xrightarrow{t,e} s'$ will be called a *mixed move*. If ϱ is a finite run and ends in some s_n with $\mathsf{loc}(s_n) \in \mathsf{Goal}$, we say that ϱ is accepting. If $s \in S$ is a configuration, we write $\mathsf{Runs}(\mathcal{A}, s)$ (respectively $\mathsf{Runs}_{\mathsf{f}}(\mathcal{A}, s)$, $\mathsf{Runs}_{\mathsf{f}}^{\mathsf{acc}}$) the set of infinite (respectively finite, finite accepting) runs that start in s. An infinite run $\varrho = s \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \ldots$ in $\mathsf{Runs}(\mathcal{A}, s)$ is said *Zeno* whenever $\sum_{i \in \mathbb{N}} t_i < \infty$.

If $\mathcal{A}$ is a timed automaton, a configuration (ℓ, v) is *valid* whenever $v \models \mathsf{Inv}(\ell)$. A timed automaton $\mathcal{A}$ is said *non-blocking* whenever for every valid configuration s of $\mathcal{A}$, there exists some delay t and some edge e, there exists some configuration s' such that $s \xrightarrow{t,e} s'$ is a mixed move of $\mathcal{A}$. Note that this is an easy condition to be checked. In the following, we will assume that timed automata are non-blocking, to avoid boring considerations, even though most developments can also be made in more general models.

Unless specifically mentioned, we assume that constants appearing in constraints of timed automata are integers, and no more rationals, as assumed in the initial definition.[3] We know, see [AD94, Lemma 4.1], that this is without loss of generality.

2.2 Expressing properties

2.2.1 Basic untimed properties

Let $\mathcal{A} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv})$ be a timed automaton. We say that (i) $\mathcal{A}$ satisfies the *reachability property* with respect to Goal if there exists a run

[2] Letting $s = (\ell, v)$, we will later write $s + t$ for the configuration $(\ell, v + t)$.

[3] We chose rational constants in the definition for convenience. It will for instance be useful in chapter 4.

$\varrho \in \mathsf{Runs}_{\mathsf{f}}^{\mathsf{acc}}(\mathcal{A}, s_0)$, and (ii) $\mathcal{A}$ satisfies the *safety property* with respect to Goal if there is no run $\varrho \in \mathsf{Runs}_{\mathsf{f}}^{\mathsf{acc}}(\mathcal{A}, s_0)$ (hence Goal is avoided). Reachability and safety properties are dual properties, and are the most basic properties we will be interested in.

An *ω-regular* property φ over AP is a language over the alphabet $\mathsf{Bool}(\mathsf{AP})$,[4] the set of Boolean combinations of atomic propositions, that can be defined by an ω-automaton $\mathcal{B}_\varphi$ (for instance with a Muller acceptance condition). Let $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, E, \mathsf{Inv}, \mathcal{L})$ be a timed automaton. An infinite run $\varrho = s \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \dots$ in $\mathsf{Runs}(\mathcal{A}, s)$ satisfies the ω-property φ whenever there is an infinite word $uu_1u_2\dots$ over alphabet $\mathsf{Bool}(\mathsf{AP})$ which is accepted by $\mathcal{B}_\varphi$, and such that $\mathcal{L}(s) \models u$ and for every $i \in \mathbb{N}_{>0}$, $\mathcal{L}(s_i) \models u_i$. We write $\varrho \models \varphi$ when the run ϱ satisfies the ω-regular property φ, and $\mathcal{A} \models \varphi$ whenever for every run $\varrho \in \mathsf{Runs}(\mathcal{A}, s_0)$, $\varrho \models \varphi$.

The *model-checking problem* asks, given a timed automaton $\mathcal{A}$ and an ω-regular property φ, whether $\mathcal{A} \models \varphi$. For complexity issues, for an ω-regular property, we will assume that the size of the input is the size of the corresponding ω-automaton.

***Remark* 2.3** Reachability and safety properties can obviously be expressed as ω-regular properties. We label Goal locations (and only those locations) with a specific atomic proposition g, and we consider the following ω-automata, where two concentric circles denote accepting states for a Büchi accepting condition, and unlabelled transitions are implicitly labelled by 'true'.

reachability property safety property

⌟

2.2.2 Classical temporal logics

In this subsection we define two classical temporal logics that are used to specify properties of systems. We then explain how we interpret those properties on timed automata.

We fix a finite set AP of atomic propositions.

[4]The set $\mathsf{Bool}(\mathsf{AP})$ is infinite, but we assume we take one representative (of minimal size) per set of equivalent formulas.

The linear-time temporal logic LTL. The syntax of LTL [Pnu77] over AP is given by the following grammar:

$$\mathsf{LTL} \ni \varphi \ ::= \ p \ \mid \ \neg\varphi \ \mid \ \varphi \vee \varphi \ \mid \ \varphi \,\mathbf{U}\, \varphi$$

where $p \in \mathsf{AP}$. The $\mathbf{U}$-modality is called the 'Until' modality.

Two semantics can be naturally defined for LTL over timed automata. The first one evaluates formulas when transitions are taken (it will be called the *pointwise semantics*), and the second one evaluates formulas continuously (it will be called the *continuous semantics*). In the following we will distinguish between the two semantics. They share rules for basic modalities, and only differ in the interpretation of the term *position*. We interpret LTL formulas over runs of a timed automaton, from some position along that run.

We fix a timed automaton $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L})$, and we let $\varrho = s \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \cdots s_{n-1} \xrightarrow{t_n,e_n} s_n \cdots$ be a finite or infinite run of $\mathcal{A}$, and ϖ be a position along ϱ. The satisfaction relation for ϱ from ϖ (written $(\varrho, \varpi) \models \varphi$ if ϱ satisfies φ from ϖ) is defined inductively as follows:

$$\begin{array}{rcl}
(\varrho,\varpi) \models p & \Leftrightarrow & p \in \mathcal{L}(\varrho_{[\varpi]}) \\
(\varrho,\varpi) \models \neg\varphi & \Leftrightarrow & (\varrho,\varpi) \not\models \varphi \\
(\varrho,\varpi) \models \varphi \vee \psi & \Leftrightarrow & (\varrho,\varpi) \models \varphi \text{ or } (\varrho,\varpi) \models \psi \\
(\varrho,\varpi) \models \varphi \,\mathbf{U}\, \psi & \Leftrightarrow & \text{there exists a position } \varpi' > \varpi \text{ along } \varrho \\
 & & \text{such that } (\varrho,\varpi') \models \psi, \text{ and} \\
 & & \text{for every position } \varpi < \varpi'' < \varpi',\ (\varrho,\varpi'') \models \varphi
\end{array}$$

where $\varrho_{[\varpi]}$ is the configuration along ϱ at position ϖ.

- In the **continuous semantics**, a position in a run ϱ is any state appearing along ϱ: it can for instance be formally defined as a pair $(i,t) \in \mathbb{N} \times \mathbb{R}_{\geqslant 0}$ where i is the index of the last edge that has been fired before that position, and t is the delay since that edge was fired. For instance for the i-th transition of the run $(\ell_{i-1}, v_{i-1}) \xrightarrow{t_i,e_i} (\ell_i, v_i)$, any state $(\ell_{i-1}, v_{i-1} + t)$ with $0 \leqslant t \leqslant t_i$ is a position of ϱ (represented as the pair $(i-1, t)$), and obviously, so is (ℓ_i, v_i) (represented as the pair $(i, 0)$). This semantics is very strong because for ϱ to satisfy $\varphi \,\mathbf{U}\, \psi$, all intermediary states of ϱ need to satisfy φ before ψ holds.

- In the **pointwise semantics**, a position in the run ϱ is one of the states s_i, represented by its index i. In this semantics, formulas are checked only right after a transition has been taken. Sometimes, the pointwise semantics is given in terms of timed words (see table 2.1). When it will be more convenient, we will used this equivalent terminology.

The initial position ϖ_0 of a run represents its initial configuration, and we write $\varrho \models \varphi$ whenever $(\varrho, \varpi_0) \models \varphi$. Let $\mathcal{A}$ be a timed automaton with initial configuration s_0, and φ be an LTL formula. We say that $\mathcal{A}$ satisfies φ over infinite (respectively finite) runs, and we write $\mathcal{A} \models \varphi$ (respectively $\mathcal{A} \models_{\mathsf{f}} \varphi$), whenever for all infinite runs $\varrho \in \mathsf{Runs}(\mathcal{A}, s_0)$ (respectively $\varrho \in \mathsf{Runs}^{\mathsf{acc}}_{\mathsf{f}}(\mathcal{A}, s_0)$), $\varrho \models \varphi$.

The pointwise semantics of LTL can be defined in terms of timed words instead of runs, and the position is then an index. Let $w = (a_0, \tau_0)(a_1, \tau_1)(a_2, \tau_2)\cdots$ be a (finite or infinite) timed word, and $i \in \mathbb{N}$. The pointwise semantics of LTL is redefined as follows:

$$\begin{array}{rcl}
(w,i) \models p & \Leftrightarrow & p \in a_i \\
(w,i) \models \neg\varphi & \Leftrightarrow & (w,i) \not\models \varphi \\
(w,i) \models \varphi \vee \psi & \Leftrightarrow & (w,i) \models \varphi \text{ or } (w,i) \models \psi \\
(w,i) \models \varphi\,\mathbf{U}\,\psi & \Leftrightarrow & \text{there exists } i' > i \\
 & & \text{such that } (w,i') \models \psi, \text{ and} \\
 & & \text{for every } i < i'' < i',\ (w,i'') \models \varphi
\end{array}$$

Given a timed automaton $\mathcal{A}$ and a run $\varrho = s \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \cdots$ in $\mathcal{A}$, we have that, in the pointwise semantics:,

$$(\varrho, i) \models \varphi \;\Leftrightarrow\; (w, i+1) \models \varphi$$

where w is the timed word $(\mathcal{L}(s), 0)(\mathcal{L}(s_1), t_1)(\mathcal{L}(s_2), t_1 + t_2)\cdots$ over the alphabet 2^{AP}. Note that this is somehow as if we had puts labels on transitions.

Table 2.1: The pointwise semantics in terms of timed words

We define some syntactic sugar for LTL: $\mathtt{true} \equiv (p \vee \neg p)$ stands for true, $\mathtt{false} \equiv (\neg\mathtt{true})$ stands for false, $(\varphi \rightarrow \psi) \equiv (\neg\varphi \vee \psi)$ is the classical implication, $\mathbf{F}\,\varphi \equiv (\mathtt{true}\,\mathbf{U}\,\varphi)$ (φ will eventually hold), $\mathbf{G}\,\varphi \equiv \neg(\mathbf{F}\,\neg\varphi)$ (φ holds everywhere), and $\mathbf{X}\,\varphi \equiv (\mathtt{false}\,\mathbf{U}\,\varphi)$ (at the next position, φ holds).

The next result plays a fundamental role in verification, and current research still improves the practical issues behind this theoretical result.

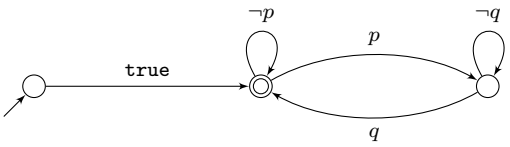
Proposition 2.4 ([WVS83, VW94, Wol00]) *In the pointwise semantics, any* LTL *formula defines an* ω*-regular property, and the size of an equivalent* ω*-automaton is exponential.*

***Remark* 2.5** The above proposition is not stated in the continuous semantics,

because we have chosen the strict semantics for the **U**-modality,[5] and due to that choice, two runs $\varrho_1 = s \xrightarrow{0,e} s_1$ and $s \xrightarrow{0.1,e} s_2$ in a timed automaton with $\mathcal{L}(s) = \{p\}$ and $\mathcal{L}(s_1) = \mathcal{L}(s_2) = \emptyset$ will not satisfy the same LTL properties in the continuous semantics. For instance, ϱ_1 does not satisfy '$\mathbf{F}\, p$' in the continuous semantics, whereas ϱ_2 does (a position along that run is labelled by p, *eg.* '$s+0.1$'). There are some subtleties in the choice of the semantics for the **U**-modality, that we will not discuss further.

⌟

***Example* 2.6** The property $\mathbf{G}\,(p \rightarrow \mathbf{F}\, q)$ mentioned in the introduction (with $p \equiv \mathsf{car.crash}$ and $q \equiv \mathsf{airbag.inflate}$) is in LTL. It is equivalent to the ω-regular property defined by the following automaton (with a Büchi accepting condition):



Note that the first transition labelled by `true` is due to the choice of the strict semantics for the **U**-modality.

⌟

The branching-time temporal logic CTL. The syntax of CTL [CE82] over AP is given by the following grammar:

$$\mathsf{CTL} \ni \varphi \ ::= \ p \ \mid \ \neg\varphi \ \mid \ \varphi \vee \varphi \ \mid \ \mathbf{E}\,(\varphi\,\mathbf{U}\,\varphi) \ \mid \ \mathbf{A}\,(\varphi\,\mathbf{U}\,\varphi)$$

where $p \in \mathsf{AP}$.

Formulas of CTL are interpreted over configurations of a timed automaton $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, E, \mathsf{Inv}, \mathcal{L})$. Given s a configuration of $\mathcal{A}$ and $\varphi \in \mathsf{CTL}$, the satisfaction relation is defined inductively as follows, writing $s \models \varphi$ whenever φ is satisfied from s in $\mathcal{A}$:

$$\begin{array}{rcl}
s \models p & \Leftrightarrow & p \in \mathcal{L}(s) \\
s \models \neg\varphi & \Leftrightarrow & s \not\models \varphi \\
s \models \varphi_1 \vee \varphi_2 & \Leftrightarrow & s \models \varphi_1 \text{ or } s \models \varphi_2 \\
s \models \mathbf{E}\,(\varphi\,\mathbf{U}\,\psi) & \Leftrightarrow & \text{there exists } \varrho \in \mathsf{Runs}(\mathcal{A}, s) \text{ and a position } \varpi \\
 & & \text{along } \varrho \text{ such that } \varrho_{[\varpi]} \models \psi, \text{ and} \\
 & & \text{for every position } \varpi_0 < \varpi' < \varpi \text{ along } \varrho,\ \varrho_{[\varpi']} \models \varphi \\
s \models \mathbf{A}\,(\varphi\,\mathbf{U}\,\psi) & \Leftrightarrow & \text{for every } \varrho \in \mathsf{Runs}(\mathcal{A}, s), \text{ there is a position } \varpi \\
 & & \text{along } \varrho \text{ such that } \varrho_{[\varpi]} \models \psi, \text{ and} \\
 & & \text{for every position } \varpi_0 < \varpi' < \varpi \text{ along } \varrho,\ \varrho_{[\varpi']} \models \varphi
\end{array}$$

[5] In the semantics of the **U**-modality, positions ϖ' and ϖ'' are taken in such a way that $\varpi < \varpi'' < \varpi'$. An alternative would be to assume that $\varpi \leqslant \varpi'' \leqslant \varpi'$ but this yields a less general definition.

where ϖ_0 still denotes the initial position of the run, and where we take the same notations as in the paragraph about LTL. Also, we make the same distinctions for the interpretation of the term position, even though this will be less fundamental. Finally, note that, as for LTL, we could have distinguished between finite and infinite runs, but that will not play any real role, hence we only consider the interpretation over infinite runs.

***Example* 2.7** We can write that the airbag inflates each time the car crashes:

$$\mathbf{A}\,\mathbf{G}\,\big(\mathsf{car.crash} \Rightarrow\ \mathbf{A}\,\mathbf{F}\,\mathsf{airbag.inflate}\big).$$

Also, we can write that we can repair the machine when it fails:

$$\mathbf{A}\,\mathbf{G}\,\big(\mathsf{failure} \Rightarrow\ \mathbf{E}\,\mathbf{F}\,\mathsf{repair}\big).$$

⌟

2.3 The region automaton abstraction

In this section, we recall the classical region automaton construction [AD90, AD94], and state some of the decidability and complexity results that can be proven, based on that construction.

2.3.1 The region equivalence

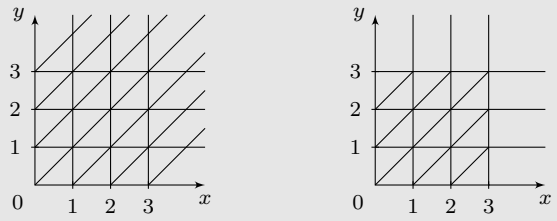
We fix a timed automaton $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L})$, and we define $M = \max\{|c| \in \mathbb{N} \mid x \sim c \text{ or } x - y \sim c \text{ constraint labelling an edge of } \mathcal{A}\}$. Given two valuations v and v' in $\mathbb{R}^X_{\geqslant 0}$, we say that they are *$M$-region equivalent*, and we write $v \cong_M v'$, whenever:

- for every clock $x \in X$, $v(x) > M$ if and only if $v'(x) > M$,
- for every clock $x \in X$, if $v(x) \leqslant M$, then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$,[6] and $v(x) = \lfloor v(x) \rfloor$ if and only if $v'(x) = \lfloor v'(x) \rfloor$,
- for every pair of clocks $(x, y) \in X^2$, for every integer $-M \leqslant c \leqslant M$, $v(x) - v(y) \leqslant c$ if and only if $v'(x) - v'(y) \leqslant c$.[7]

[6] $\lfloor \alpha \rfloor$ denotes the integral part of α.

[7] If the timed automaton is diagonal-free, we can change this condition into: "if $v(x) \leqslant M$ and $v(y) \leqslant M$ then $(\{v(x)\} \leqslant \{v(y)\}$ if and only if $\{v'(x)\} \leqslant \{v'(y)\})$", where $\{\alpha\}$ denotes the fractional part of α.

We will give the partitions yielded by the equivalence relation $\cong_3$ when there are two clocks, called x and y, in the general case (on the left) and in the diagonal-free case (on the right).



A region is one piece of the partition. In the following, we may represent the abstract behaviours in the region partition as follows: while time elapses, regions are visited following the diagonal, hence the time successor of a triangular region is a flat region, whose successor is a triangular region, *etc.* When resetting a clock, we just project the region onto one of the axes. This is illustrated by the following picture, that we will reuse later.

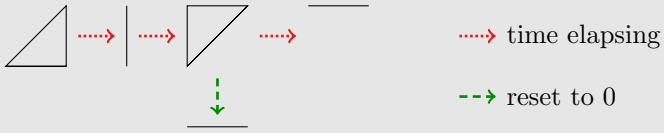


Table 2.2: The region equivalence

The equivalence $\cong_M$ has finite index, and an equivalence class of $\cong_M$ is called a *region*. We give an example of region equivalence in table 2.2. It is rather tedious (but not really difficult) to prove that the relation $\equiv_{\mathcal{A}}$ defined by

$$(\ell, v) \equiv_{\mathcal{A}} (\ell', v') \quad \Leftrightarrow \quad \begin{cases} \ell = \ell' \\ v \cong_M v' \end{cases}$$

is a *time-abstract bisimulation* (see table 2.3). If s is a configuration of $\mathcal{A}$, we write $[s]_{\mathcal{A}}$ the equivalence class of $\mathcal{A}$ with respect to the equivalence $\equiv_{\mathcal{A}}$.

2.3.2 The region automaton

The *region automaton* of the timed automaton $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L})$ is the (labelled) finite automaton $\Gamma(\mathcal{A}) = (\mathsf{AP}, Q, q_0, Q_f, T, \lambda)$ such that:

Let $\mathcal{T}_{\mathcal{A}} = (S, s_0, \rightarrow)$ be the timed transition system associated with the timed automaton $\mathcal{A}$. A relation $\mathfrak{R} \subseteq (S \times S)$ is a *time-abstract bisimulation* whenever it is a symmetric relation satisfying the two following conditions:

- if $s_1 \xrightarrow{e} s_1'$ and $(s_1, s_2) \in \mathfrak{R}$, there exists $s_2' \in S$ such that $s_2 \xrightarrow{e} s_2'$ and $(s_1', s_2') \in \mathfrak{R}$;
- if $s_1 \xrightarrow{t_1} s_1 + t_1$ for some $t_1 \in \mathbb{R}_{\geqslant 0}$ and $(s_1, s_2) \in \mathfrak{R}$, there exists $t_2 \in \mathbb{R}_{\geqslant 0}$ such that $s_2 \xrightarrow{t_2} s_2 + t_2$ and $(s_1 + t_1, s_2 + t_2) \in \mathfrak{R}$.

This is a simplified definition (for instance, usually, we consider actions labelling edges, not the edges themselves), but it is convenient (and sufficient) for our purpose.

Table 2.3: Time-abstract bisimulation

- $Q = (L \times \mathbb{R}^X_{\geqslant 0}) / \equiv_{\mathcal{A}}$,
- $q_0 = [s_0]_{\mathcal{A}}$ (where $s_0 = (\ell_0, \mathbf{0}_X)$),
- $Q_f = \big\{ [s]_{\mathcal{A}} \mid \mathsf{loc}(s) \in \mathsf{Goal} \big\}$,
- $q \xrightarrow{e} q'$ if there is a mixed move $s \xrightarrow{t,e} s'$ in $\mathcal{T}_{\mathcal{A}}$ with $q = [s]_{\mathcal{A}}$ and $q' = [s']_{\mathcal{A}}$,[8]
- $\lambda(q) = \mathcal{L}(s)$ if $q = [s]_{\mathcal{A}}$.

We extend the notions of runs, and the various notations to finite automata in a straightforward way. Furthermore, we say that a transition $q \xrightarrow{e} q'$ of $\Gamma(\mathcal{A})$ is *implicitly labelled by a constraint* g whenever $[\![g]\!] = \{\mathsf{val}(s + t) \in \mathbb{R}^X_{\geqslant 0} \mid s \xrightarrow{t,e} s' \text{ is a mixed move in } \mathcal{T}_{\mathcal{A}}\}$.

Thanks to the time-abstract bisimulation property mentioned before, it is not difficult to prove the following proposition.

Proposition 2.8 *Let $\mathcal{A}$ be a timed automaton with initial configuration s_0.*

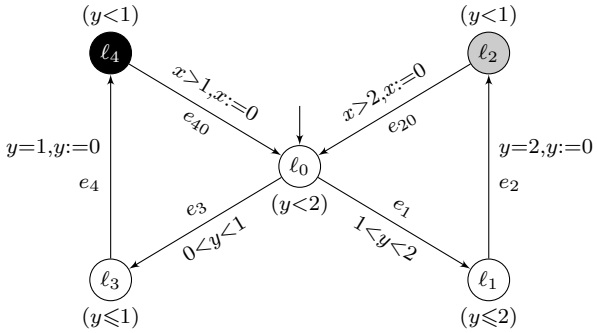
- *There is a finite run in $\mathsf{Runs}^{\mathsf{acc}}_{\mathsf{f}}(\mathcal{A}, s_0)$ if and only if there is a finite run in $\mathsf{Runs}^{\mathsf{acc}}_{\mathsf{f}}(\Gamma(\mathcal{A}), [s_0]_{\mathcal{A}})$.*
- *There is a run in $\mathsf{Runs}(\mathcal{A}, s_0)$ satisfying an ω-regular property φ if and only if there is a run in $\mathsf{Runs}(\Gamma(\mathcal{A}), [s_0]_{\mathcal{A}})$ satisfying φ.*

[8] Because $\equiv_{\mathcal{A}}$ is a time-abstract bisimulation, this condition is equivalent to "for every s such that $[s]_{\mathcal{A}} = q$, there is a mixed move $s \xrightarrow{t,e} s'$ with $[s']_{\mathcal{A}} = q'$".

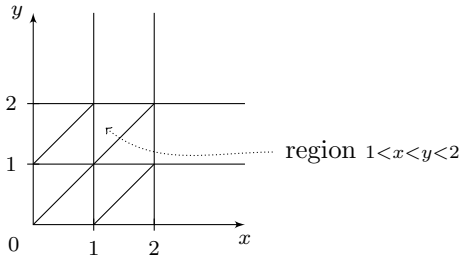
- *All runs in* $\mathsf{Runs}(\mathcal{A}, s_0)$ *satisfy an* ω*-regular property* φ *if and only if all runs in* $\mathsf{Runs}(\Gamma(\mathcal{A}), [s_0]_{\mathcal{A}})$ *satisfy* φ*.*

***Remark* 2.9** Using language-theoretic terms (*cf.* [AD94]), the timed automaton $\mathcal{A}$ accepts a timed word $w = (a_1, \tau_1)(a_2, \tau_2)\cdots(a_n, \tau_n)$ if and only if the finite automaton $\Gamma(\mathcal{A})$ accepts the (finite untimed) word $\mathsf{untime}(w) = a_1 a_2 \cdots a_n$. ⌟

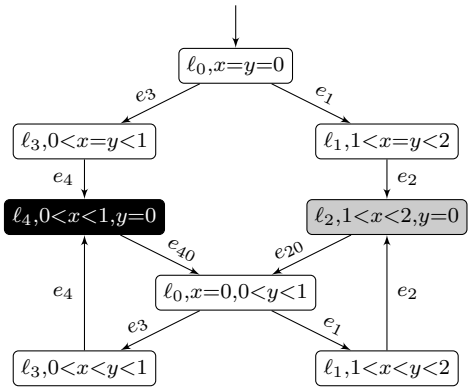
***Example* 2.10** We illustrate the construction on the timed automaton $\mathcal{A}$:



The set of regions for that (diagonal-free) automaton is:



The region automaton $\Gamma(\mathcal{A})$ of $\mathcal{A}$ is the following finite automaton:



It is easy to match every run in $\mathcal{A}$ onto $\Gamma(\mathcal{A})$, and *vice-versa*. Furthermore the existence of a run satisfying the LTL property '$\mathbf{G}\,\mathbf{F}$ black $\wedge$ $\mathbf{G}\,\mathbf{F}$ grey' in $\Gamma(\mathcal{A})$ implies the existence of a run satisfying that property in $\mathcal{A}$. ⌟

2.3.3 Decidability and complexity results

Now it is not difficult to compute that the number of regions of a timed automaton $\mathcal{A}$ is bounded by $(2M+2)^{(|X|+1)^2}$ in the general case, and by $2^{|X|} \cdot |X|! \cdot (2M+2)^{|X|}$ in the diagonal-free case. In both cases, there are thus exponentially many regions. However, in general the finite automaton $\Gamma(\mathcal{A})$ needs not be computed, and non-deterministic algorithms can be used to check basic reachability, safety or ω-regular properties. Hence we get the following result, which has been and is still fundamental for the development of verification techniques for timed systems.

Theorem 2.11 ([AD90, AD94]) *Model-checking reachability, safety, or ω-regular properties in timed automata is decidable, and* PSPACE*-complete.*

The PSPACE-hardness can be obtained by simulating a linearly-bounded Turing machine. This is not our purpose to give more details, we thus better refer to [AD94, AL02].

More generally, we can mix classical complexity results for the model-checking of LTL and CTL over finite automata [CES83, SC85] and the above region automaton construction to get the following theorem.

Theorem 2.12 *Model-checking* LTL *or* CTL *properties in timed automata is* PSPACE*-complete.*

Note that even though the two model-checking problems belong to the same complexity class, the model-checking of LTL is harder, because it is both exponential in the size of the system and in the size of the formula, whereas the model-checking of CTL is polynomial in the size of the formula and exponential in the size of the system.

Chapter 3

Reachability analysis in timed systems

3.1 Introduction

By their very definition timed automata describe (uncountable) infinite state-spaces. Thus, algorithmic verification relies on the existence of exact finite abstractions. In the original work by Alur and Dill, the so-called region automaton construction provides such an abstraction. However, whereas well-suited for establishing decidability of problems related to timed automata, the region automaton is highly impractical from a tool implementation point-of-view. Instead, most real-time verification tools (like CMC [1] [LL98], Kronos [2] [DOTY96], and Uppaal [3] [BDL+06]) apply abstractions based on so-called zones, which in practice provide much coarser abstractions.

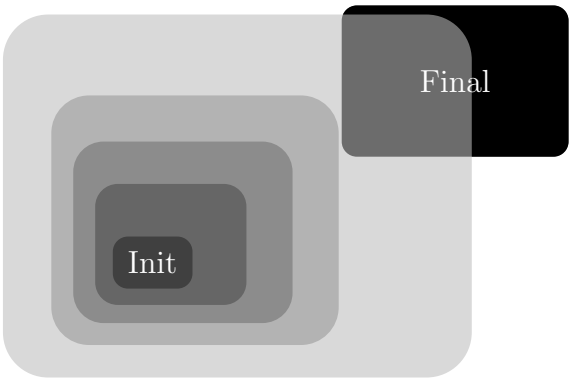
In this chapter, we review the most basic methods that can be used for analysing timed automata. We focus on simple reachability properties (or equivalently on basic safety properties). We first describe the two forward and backward paradigms, that are not specific to the analysis of timed systems, and then apply these two methods to the timed automata framework. We discuss in particular the termination and the correctness of the two approaches. For the case of the forward analysis, abstraction operators need to be used to ensure termination of the computation, we thus describe the abstraction operator which is commonly used, and finally present several improvements thereof.

[1] http://www.lsv.ens-cachan.fr/~fl/cmcweb.html/
[2] http://www-verimag.imag.fr/TEMPORISE/kronos/
[3] http://www.uppaal.com/

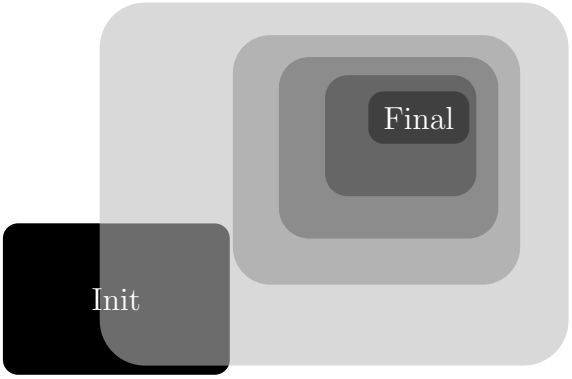
3.2 Checking reachability properties: two general methods

There are two main approaches for checking reachability (or safety) properties in systems (not only timed systems, but all kinds of systems). We describe these two approaches shortly and apply them to timed automata in the next section.

- *Forward analysis.* The general idea is to compute configurations which are reachable from the initial configuration within 1 step, 2 steps, *etc.* until final (or goal) configurations are computed, or until the computation terminates. The forward analysis computation can be schematized as below.



- *Backward analysis.* The general idea is to compute configurations from which we can reach final configurations within 1 step, 2 steps, *etc.* until the initial configuration is computed, or until the computation terminates. The backward analysis computation can be represented as below.



These two generic approaches are used in many contexts, including the analysis of models like counter machines, hybrid systems, *etc.* Of course, given a class of systems, specific techniques (*eg.* abstractions, widening operations, *etc.*) can be used for improving the computations. We will now focus on timed automata and explain how these two approaches can be implemented in that framework.

3.3 Reachability analysis in timed automata: the zone symbolic representation

Timed automata have infinitely (and even uncountably) many configurations, it is thus necessary to use symbolic representations for doing the computations. For the discussion which follows we fix a timed automaton $\mathcal{A} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv})$. Given an edge $e = \left(\ell \xrightarrow{g,a,Y} \ell'\right)$ of $\mathcal{A}$, we need to be able to compute its effects on a set of valuations. More precisely, if W is a set of valuations, we define the two following sets of valuations:

$$\begin{cases} \mathsf{Post}_e(W) &= \{v' \in \mathbb{R}^X_{\geqslant 0} \mid \exists v \in W\ \exists t \in \mathbb{R}_{\geqslant 0} \text{ such that } v + t \models g \\ & \qquad\qquad \text{and } v' = [Y \leftarrow 0](v + t)\} \\ \mathsf{Pre}_e(W) &= \{v \in \mathbb{R}^X_{\geqslant 0} \mid \exists v' \in W\ \exists t \in \mathbb{R}_{\geqslant 0} \text{ such that } v + t \models g \\ & \qquad\qquad \text{and } [Y \leftarrow 0](v + t) = v'\} \end{cases}$$

A valuation v' is in $\mathsf{Post}_e(W)$ whenever there exists some valuation $v \in W$ and some $t \in \mathbb{R}_{\geqslant 0}$ such that $(\ell, v) \xrightarrow{t,e} (\ell', v')$ is a mixed move in $\mathcal{T}_\mathcal{A}$. Similarly a valuation v is in $\mathsf{Pre}_e(W)$ whenever there exists some valuation $v' \in W$ and some $t \in \mathbb{R}_{\geqslant 0}$ such that $(\ell, v) \xrightarrow{t,e} (\ell', v')$ is a mixed move in $\mathcal{T}_\mathcal{A}$.

It is worth noticing that if W is a *zone*, *i.e.*, a set of valuations defined by a general clock constraint, then for every transition e of $\mathcal{A}$, $\mathsf{Post}_e(W)$ and $\mathsf{Pre}_e(W)$ are both zones. For analysing timed automata, zones are the most basic and commonly used *symbolic representation*.

In the following, we will need to decompose the computation of Post_e and Pre_e in several simpler steps, hence we define the following operations on zones (or more generally on sets of valuations):

- Future of W: $\overrightarrow{W} = \{v + t \mid v \in W \text{ and } t \in \mathbb{R}_{\geqslant 0}\}$
- Past of W: $\overleftarrow{W} = \{v - t \mid v \in W \text{ and } t \in \mathbb{R}_{\geqslant 0}\}$
- Intersection of W and W': $W \cap W' = \{v \mid v \in W \text{ and } v \in W'\}$

- Reset to zero of W with respect to the set of clocks Y:
$$[Y \leftarrow 0]W = \{[Y \leftarrow 0]v \mid v \in W\}$$
- Inverse reset to zero of W with respect to the set of clocks Y:
$$[Y \leftarrow 0]^{-1}W = \{v \mid [Y \leftarrow 0]v \in W\}$$

It can be shown that all those elementary operations preserve zones, and moreover, they allow to express the Post_e and Pre_e operators:

$$\begin{cases} \mathsf{Post}_e(W) &= [Y_e \leftarrow 0](\overrightarrow{W} \cap [\![g_e]\!]) \\ \mathsf{Pre}_e(W) &= \overleftarrow{[Y_e \leftarrow 0]^{-1}(W \cap [\![Y_e = 0]\!]) \cap [\![g_e]\!]} \end{cases}$$

3.4 The DBM data structure

The most common data structure for representing zones is the so-called DBM data structure. This data structure has been first introduced in [BM83] and then set in the framework of timed automata in [Dil90]. Several presentations of this data structure can be found in the literature, for example in [CGP99, Ben02, Bou04].

A *difference bound matrix*, we shall write DBM for short, for a set $X = \{x_1, \ldots, x_n\}$ of n clocks is an $(n+1)$-square matrix of pairs

$$(\prec, m) \in \mathbb{V} = (\{<, \leqslant\} \times \mathbb{Z}) \cup \{(<, \infty)\}.$$

A DBM $M = (\prec_{i,j}, m_{i,j})_{0 \leqslant i,j \leqslant n}$ defines the following subset of $\mathbb{R}^X_{\geqslant 0}$ (if $v \in \mathbb{R}^X_{\geqslant 0}$, $\overline{v}$ is the 'canonical' valuation over $X \cup \{x_0\}$ — where x_0 is a fresh clock — such that $\overline{v}(x) = v(x)$ for every $x \in X$, and $\overline{v}(x_0) = 0$; In the following we may write v instead of $\overline{v}$):

$$\{v : X \to \mathbb{R}_{\geqslant 0} \mid \forall\ 0 \leqslant i, j \leqslant n,\ \overline{v}(x_i) - \overline{v}(x_j) \prec_{i,j} m_{i,j}\}$$

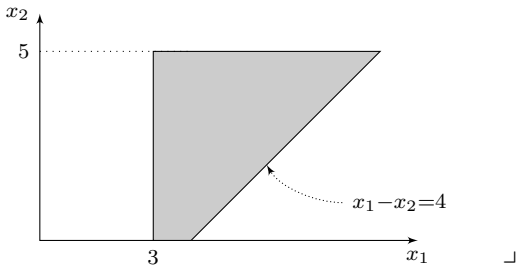
where $\gamma < \infty$ simply means that $\gamma \in \mathbb{R}_{\geqslant 0}$ (without any constraint on γ). This subset of $\mathbb{R}^X_{\geqslant 0}$ is a zone and will be denoted by $[\![M]\!]$. To simplify the notations, we now assume that all constraints are non-strict, so that coefficients of DBMs will simply be elements of $\mathbb{Z} \cup \{\infty\}$.

***Example* 3.1** We consider the zone over the set of clocks $X = \{x_1, x_2\}$ defined by the general clock constraint

$$(x_1 \geqslant 3)\ \wedge\ (x_2 \leqslant 5)\ \wedge\ (x_1 - x_2 \leqslant 4).$$

This zone, depicted on the next picture on the right, can be represented by the DBM on the left.

$$\begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} \infty & -3 & \infty \\ \infty & \infty & 4 \\ 5 & \infty & \infty \end{pmatrix} \end{array}$$



A zone can have several representations using DBMs. For example, the zone of the previous example can equivalently be represented by the DBM

$$\begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix} \end{array}$$

However there is a normal form for DBMs, which tightens all possible constraints. This can be done using a Floyd-Warshall algorithm on the matrix (viewed as the adjacency matrix of a weighted graph). A zone has a unique representation as a DBM in normal form. Tests like emptiness checks or comparisons of zones can be done syntactically on the DBMs in normal form. For example, a zone Z is included in a zone Z' if the DBM in normal form representing Z is smaller than the DBM in normal form representing Z'.[4] Finally all operations on zones described in section 3.3 can easily be done using DBMs, details can be found in all previously mentioned papers on DBMs.

Let us just mention that the DBM data structure is the most basic data structure which is used for analysing timed systems, some more involved BDD-like data structures can also be used, for example CDDs (which stands for 'Clock Difference Diagrams') [LPWY99], or more recently federations [DHGP04, Dav05].

3.5 Backward analysis

We first focus on the backward analysis computation, which will surprisingly turn out to be the simplest to analyse. We fix a timed automaton $\mathcal{A} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv})$.

[4]The order is taken coefficient by coefficient.

3.5.1 Backward symbolic transition system

The *backward symbolic transition system* associated with $\mathcal{A}$ is denoted by '$\Leftarrow$' and is defined inductively as follows:

$$\frac{e = \left(\ell_1 \xrightarrow{g,a,Y} \ell_2\right) \in E \qquad W_1 = \mathsf{Pre}_e(W_2)}{(\ell_2, W_2) \Leftarrow (\ell_1, W_1)}$$

Obviously, if we write $\Leftarrow^*$ for the reflexive and transitive closure of $\Leftarrow$, we have that $(\ell', W') \Leftarrow^* (\ell, W)$ if and only if for every $v \in W$, there exists $v' \in W'$ and a run in $\mathcal{A}$ from (ℓ, v) to (ℓ', v').

The backward analysis algorithm then consists in computing iteratively the following sets of symbolic configurations:

$$\begin{array}{rcl}
\mathcal{S}^{\mathsf{b}}_0 & = & \{(\ell, \mathbb{R}^X_{\geqslant 0}) \mid \ell \in \mathsf{Goal}\} \\
\mathcal{S}^{\mathsf{b}}_1 & = & \mathcal{S}^{\mathsf{b}}_0 \cup \{(\ell, W) \mid \exists (\ell', W') \in \mathcal{S}^{\mathsf{b}}_0 \text{ such that } (\ell', W') \Leftarrow (\ell, W)\} \\
& \vdots & \\
\mathcal{S}^{\mathsf{b}}_{p+1} & = & \mathcal{S}^{\mathsf{b}}_p \cup \{(\ell, W) \mid \exists (\ell', W') \in \mathcal{S}^{\mathsf{b}}_p \text{ such that } (\ell', W') \Leftarrow (\ell', W')\} \\
& \vdots &
\end{array}$$

until either (i) the computation stabilizes, or (ii) a symbolic state is computed, which contains the initial configuration of $\mathcal{A}$. To help event (i) happen, it is possible to add the following inclusion check: if $(\ell, W) \in \mathcal{S}^{\mathsf{b}}_{p+1}$ and if there exists $(\ell, W') \in \mathcal{S}^{\mathsf{b}}_p$ such that $W \subseteq W'$ (or even if there exist $(\ell, W_i) \in \mathcal{S}^{\mathsf{b}}_p$ for finitely many i's such that $W \subseteq \bigcup_i W_i$), then do not include (ℓ, W) in $\mathcal{S}^{\mathsf{b}}_{p+1}$. The procedure answers 'Yes' in case (ii) and 'No' in case $(i) \wedge \neg(ii)$.

3.5.2 Termination and correctness

The backward analysis computation enjoys the following nice property, which can be seen as a consequence of the correctness of the backward analysis algorithm for TCTL [HNSY94]. However, we will give below a simple and direct proof of that result.

Theorem 3.2 *The backward computation terminates and is correct with respect to reachability properties.*[5]

[5] *I.e.* Goal is reachable if and only if it is declared as reachable by the backward computation.

Correctness is immediate as the computation is *exact* (as opposed to over-(or under-)approximate). The inclusion check does not cause any trouble as any state reachable from a configuration belonging to a symbolic state that is not added due to that test is actually already reachable from the previous symbolic state that was computed.

Termination needs some additional argument, that we sketch here. Assume that R_i's (for $1 \leqslant i \leqslant p$) are regions of $\mathcal{A}$ (as defined in page 23), then:

- $\overleftarrow{\bigcup_{i=1}^{p}} R_i$ is a finite union of regions;
- $[Y \leftarrow 0]^{-1}\left(\bigcup_{i=1}^{p} R_i\right)$ is a finite union of regions (for any set of clocks Y);
- $g \cap \left(\bigcup_{i=1}^{p} R_i\right)$ is a finite union of regions if g is a clock constraint of $\mathcal{A}$.

These properties altogether imply that each of the symbolic configurations (ℓ, W) that are added to $\mathcal{S}_i^{\mathsf{b}}$ is such that W is a finite union of regions. As there are finitely many regions, the sequence $(\mathcal{S}_i^{\mathsf{b}})_{i \geqslant 0}$ stabilizes, hence the termination of the backward computation.

Backward analysis may appear as an accurate method for analysing timed automata, but in practice, some tools (like Uppaal) prefer using a forward analysis computation. One of the reasons comes from the use of (bounded) integer variables that are really helpful for modelling real systems. Backward analysis is then not suitable for dealing with arithmetical operations: for example if we know in which interval lies the variable i and if we know that i is assigned the value $j.k + \ell.m$, it is not easy to compute the possible values of variables j, k, ℓ, m (apart from listing all possible tuples of values). For this kind of operations, forward analysis is much more suitable.

3.6 Forward analysis

In this section we focus on the forward analysis computation, which will require the development of abstractions, and more effort for proving its correctness. We fix a timed automaton $\mathcal{A} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv})$.

3.6.1 Forward symbolic transition system

The *forward symbolic transition system* associated with $\mathcal{A}$ is denoted by '$\Rightarrow$' and is defined inductively as follows:

$$\frac{e = \left(\ell_1 \xrightarrow{g,a,Y:=0} \ell_2\right) \in E \qquad W_2 = \mathsf{Post}_e(W_1)}{(\ell_1, W_1) \Rightarrow (\ell_2, W_2)}$$

Obviously, if we write $\Rightarrow^*$ for the reflexive and transitive closure of $\Rightarrow$, we have that $(\ell, W) \Rightarrow^* (\ell', W')$ if and only if for every $v' \in W'$, there exists $v \in W$ and a run in $\mathcal{A}$ from (ℓ, v) to (ℓ', v').

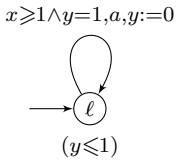
The forward analysis computation then consists in computing iteratively the following sets of symbolic configurations:

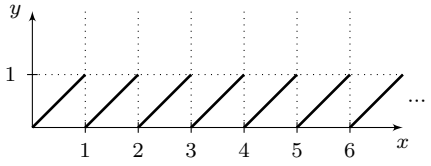
$$\begin{array}{rcl}
\mathcal{S}_0^{\mathsf{f}} & = & \{(\ell_0, \mathbf{0}_X)\} \\
\mathcal{S}_1^{\mathsf{f}} & = & \mathcal{S}_0^{\mathsf{f}} \cup \{(\ell', W') \mid \exists (\ell, W) \in \mathcal{S}_0^{\mathsf{f}} \text{ such that } (\ell, W) \Rightarrow (\ell', W')\} \\
 & \vdots & \\
\mathcal{S}_{p+1}^{\mathsf{f}} & = & \mathcal{S}_p^{\mathsf{f}} \cup \{(\ell', W') \mid \exists (\ell, W) \in \mathcal{S}_p^{\mathsf{f}} \text{ such that } (\ell, W) \Rightarrow (\ell', W')\} \\
 & \vdots &
\end{array}$$

until either (i) the computation stabilizes, or (ii) a symbolic state is computed, which contains a final configuration of the timed automaton (*i.e.*, a configuration of the form (f, v) with $f \in F$). To help event (i) happen, it is possible to add an inclusion check, as in the backward analysis computation. The procedure answers 'Yes' in case (ii) and 'No' in case $(i) \wedge \neg(ii)$.

3.6.2 Discussion on the termination and correctness

The forward analysis gives a correct answer, but it may not terminate. An example of automaton in which the forward computation does not terminate is given below. The zones that are computed by the above procedure are represented on the right part of the figure, and it is easy to check that the computation will never terminate.





To overcome this problem, it is necessary to use some abstraction operators. Several have been proposed in [DT98]: for instance, if Z and Z' are computed for the location ℓ, they are replaced by the smallest zone containing both Z and Z'; this approximation is called the *convex-hull* abstraction[6], it does not ensure termination and is only semi-correct, in the sense that a location announced as reachable might not be reachable (the convex-hull abstraction is an over-approximation). The most interesting abstraction studied in this paper is the *extrapolation* operator. We will present it now, but we first need to formalize a little more the forward analysis procedure. We follow the lines of [BBFL03, BBLP04] and define (abstract) symbolic transition systems.

3.6.3 Abstract forward symbolic transition systems

Let $\mathfrak{a}$ be an abstraction operator (possibly partially) defined on the sets of valuations ($\mathfrak{a}$ associates to sets of valuations sets of valuations). We define the *abstract forward symbolic transition system* '$\Rightarrow_{\mathfrak{a}}$' in the following way:

$$\frac{(\ell, W) \Rightarrow (\ell', W') \qquad W = \mathfrak{a}(W)}{(\ell, W) \Rightarrow_{\mathfrak{a}} (\ell', \mathfrak{a}(W'))}$$

This transition system gives naturally rise to the following forward computation in $\mathcal{A}$.

$$\begin{array}{rcl}
\mathcal{S}_0^{\mathsf{f},\mathfrak{a}} & = & \{(\ell_0, \mathfrak{a}(\{\mathbf{0}_X\}))\} \\
\mathcal{S}_1^{\mathsf{f},\mathfrak{a}} & = & \mathcal{S}_0^{\mathsf{f},\mathfrak{a}} \cup \{(\ell', W') \mid \exists (\ell, W) \in \mathcal{S}_0^{\mathsf{f},\mathfrak{a}} \text{ such that } (\ell, W) \Rightarrow_{\mathfrak{a}} (\ell', W')\} \\
& \vdots & \\
\mathcal{S}_{p+1}^{\mathsf{f},\mathfrak{a}} & = & \mathcal{S}_p^{\mathsf{f},\mathfrak{a}} \cup \{(\ell', W') \mid \exists (\ell, W) \in \mathcal{S}_p^{\mathsf{f},\mathfrak{a}} \text{ such that } (\ell, W) \Rightarrow_{\mathfrak{a}} (\ell', W')\} \\
& \vdots &
\end{array}$$

with the same halting conditions (and inclusion checks) as previously.

3.6.4 Soundness criteria

The abstraction operator $\mathfrak{a}$ is said *correct* with respect to reachability properties in $\mathcal{A}$ whenever the following holds:

$$\begin{array}{c}
\text{if } (\ell_0, \mathfrak{a}(\{\mathbf{0}_X\})) \Rightarrow_{\mathfrak{a}}^* (\ell, W) \text{ then there exists a run} \\
(\ell_0, \mathbf{0}_X) \rightarrow^* (\ell, v) \text{ with } v \in W \text{ in } \mathcal{A}
\end{array}$$

[6] It is a language abuse, because it is not really the convex hull of the two zones, but it is the smallest zone containing the convex-hull of the two zones.

The abstraction operator $\mathfrak{a}$ is said *complete* with respect to reachability properties whenever the following holds in $\mathcal{A}$:

$$\begin{array}{c}\text{if } (\ell_0, \mathbf{0}_X) \to^* (\ell, v) \text{ is a run in } \mathcal{A} \text{ then} \\ (\ell_0, \mathfrak{a}(\{\mathbf{0}_X\})) \Rightarrow_{\mathfrak{a}}^* (\ell, W) \text{ for some } W \text{ with } v \in W\end{array}$$

***Remark* 3.3** Note that these two notions could be generalized to more general properties than reachability properties, but we follow our lines and concentrate on reachability properties. ⌟

Our aim is to define abstraction operators $\mathfrak{a}$ such that the four following properties hold:

(Finiteness) $\{\mathfrak{a}(W) \mid \mathfrak{a} \text{ defined on } W\}$ is finite
(this ensures termination of the "abstract" forward computation)

(Correctness) $\mathfrak{a}$ is correct with respect to reachability

(Completeness) $\mathfrak{a}$ is complete with respect to reachability

(Effectiveness) $\mathfrak{a}$ is "effective"

The three first properties are properly defined, the last one is more informal. The effectiveness criterion expresses that the abstraction has to be easily computable. In timed automata literature this is most of the time interpreted as "$\mathfrak{a}$ has to be defined for all zones and $\mathfrak{a}(Z)$ has to be a zone when Z is a zone". Note that other effectiveness criteria could be proposed, but that is the one we choose here.

3.6.5 The extrapolation operator

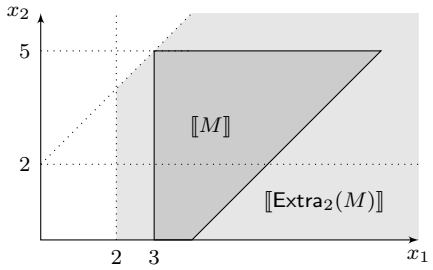
The abstraction operator which is commonly used is called *extrapolation*, and sometimes *normalization* [Ben02] or *approximation* [Bou04]. We will note it here Extra_k, it is defined up to a constant k as follows: if Z is a zone, $\mathsf{Extra}_k(Z)$ is the smallest k-bounded zone[7] which contains Z. This operation is well-defined on DBMs: if M is a DBM in normal form representing Z, a DBM representing $\mathsf{Extra}_k(Z)$ is obtained from M where each coefficient $(\prec; m)$ with $m < -k$ is replaced by $(<; -k)$ and all coefficients $(\prec; m)$ with $m > k$ is replaced by $(<; \infty)$, all other coefficients are unchanged. We write $\mathsf{Extra}_k(M)$ for this transformed DBM: it holds that $[\![\mathsf{Extra}_k(M)]\!] = \mathsf{Extra}_k([\![M]\!])$.

[7] A *k-bounded zone* is a zone defined by a k-bounded clock constraint.

***Example* 3.4** Consider again the zone introduced in example 3.1. As we have already mentioned, it can be represented by the DBM in normal form on the left and its 2-extrapolation is the DBM on the right (where we again do not mention the comparison operators):

$$M = \begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix} \quad \text{and} \quad \mathsf{Extra}_2(M) = \begin{pmatrix} 0 & -2 & 0 \\ +\infty & 0 & +\infty \\ +\infty & 2 & 0 \end{pmatrix}$$

They are both represented on the picture below.



⌟

Obviously,

- Extra_k is a finite abstraction operator because there are finitely many DBMs whose coefficients are either $(<; \infty)$ or some $(\prec; m)$ with $\prec \in \{<; \leqslant\}$ and $-k \leqslant m \leqslant k$;
- the computation of Extra_k is effective and can easily be done using DBMs;
- Extra_k is a complete abstraction with respect to reachability because for every zone Z, $Z \subseteq \mathsf{Extra}_k(Z)$.

The only point that needs to be carefully studied is the correctness of Extra_k: of course, not all constants k yield correctness, but we have to choose such a constant k carefully, so that the abstraction operator be correct with respect to reachability properties. We now discuss in details this important aspect.

3.6.6 Correctness of the extrapolation operator

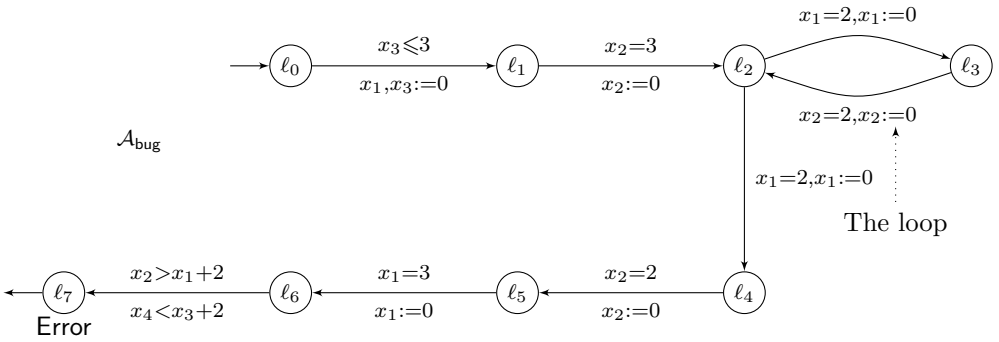
Before 2003, several attempts of proofs of the following theorem can be found in the literature, but they appear to be all incomplete or/and incorrect. We have thus written a complete and detailed correctness proof (in [Bou03, Bou04] and then in [BBFL03]), in the restricted context of diagonal-free timed automata. We will not detail the proof here, which is actually rather

technical. The main stream of the proof is to prove that any zone computed by the abstract forward transition system does not intersect a region that does not intersect a real reachable zone in the non-abstract forward transition system.

☞ **Theorem 3.5 ([Bou03, Bou04])** *Let $\mathcal{A}$ be a **diagonal-free** timed automaton. Take k the maximal constant appearing in the constraints of $\mathcal{A}$. Then Extra_k is a correct abstraction with respect to reachability properties in $\mathcal{A}$.*

***Remark* 3.6** Theorem 3.5 can be refined, and a refined extrapolation operator can be used, which uses one constant k_x per clock $x \in X$, instead of one global constant k for all the clocks. Writing K for the mapping $x \mapsto k_x$, we define the extrapolation operator Extra_K that transforms any DBM $M = (\prec_{i,j}; m_{i,j})_{0 \leqslant i,j \leqslant n}$ by replacing coefficient $(\prec_{i,j}; m_{i,j})$ by $(<; \infty)$ if $m_{i,j} > K(x_i)$, and by $(<; -k_j)$ if $m_{i,j} < -K(x_j)$. Theorem 3.5 also holds for Extra_K when for every clock x, $K(x)$ is the maximal constant to which x is compared in $\mathcal{A}$. ⌟

Surprisingly this theorem does not extend to timed automata with general clock constraints. Indeed, consider the timed automaton $\mathcal{A}_{\mathsf{bug}}$ depicted below. For every integer k, the extrapolation operator Extra_k is not correct with respect to reachability properties for $\mathcal{A}_{\mathsf{bug}}$. One can even also prove that, for automaton $\mathcal{A}_{\mathsf{bug}}$, there is no abstraction operator Abs satisfying the four above-mentioned criteria (finiteness, correctness, completeness and effectiveness). Some details are given in table 3.1.



☞ **Proposition 3.7 ([Bou03, Bou04])** *Consider the timed automaton $\mathcal{A}_{bug}$ defined before. For every integer k, Extra_k is not a correct abstraction with respect to reachability properties in $\mathcal{A}_{bug}$. Furthermore, there is no abstraction operator $\mathfrak{a}$ that over-approximates zones, and that can be finite, effective and correct with respect to reachability properties in $\mathcal{A}_{bug}$.*

We explain the problem with automaton $\mathcal{A}_{\mathsf{bug}}$. The zone Z_α which is computed by a forward analysis when reaching the location 'Error' after having taken α times the loop is defined by the constraints below (on the left). Fixing an integer k, taking α large enough the extrapolated zone is also described below (on the right).

$$Z_\alpha : \begin{cases} 1 \leqslant x_2 - x_1 \leqslant 3 \\ 1 \leqslant x_4 - x_3 \leqslant 3 \\ x_3 - x_1 = 2\alpha + 5 \\ x_4 - x_2 = 2\alpha + 5 \end{cases} \qquad \mathsf{Extra}_k(Z_\alpha) : \begin{cases} 1 \leqslant x_2 - x_1 \leqslant 3 \\ 1 \leqslant x_4 - x_3 \leqslant 3 \\ x_3 - x_2 > k \end{cases}$$

Note that if $v \in Z_\alpha$, then in particular $v(x_2) - v(x_1) = v(x_4) - v(x_3)$. On the other hand, there are valuations $v \in \mathsf{Extra}_k(Z_\alpha)$ such that $v(x_2) - v(x_1) \neq v(x_4) - v(x_3)$. Obviously, the zone Z_α does not intersect the constraint $x_2 - x_1 > 2 \wedge x_4 - x_3 < 2$, which implies that the location 'Error' is not reachable. However, $\mathsf{Extra}_k(Z_\alpha)$ intersects the constraint $x_2 - x_1 > 2 \wedge x_4 - x_3 < 2$ (for α large enough), which implies that the location 'Error' is computed as reachable by the abstract forward analysis that uses the abstraction operator Extra_k (for any integer k). The problem with automaton $\mathcal{A}_{\mathsf{bug}}$ comes from the use of diagonal constraints on the transition leading to location 'Error'.

Table 3.1: Timed automaton $\mathcal{A}_{\mathsf{bug}}$ makes the abstract forward analysis fail

Note however that for timed automata with three clocks (and possibly diagonal constraints), it is possible to find a constant k such that Extra_k is correct with respect to reachability properties (the constant k may however be larger than the maximal constant appearing in a constraint of the automaton) [Bou04]. In the general case (more than three clocks), a way of handling diagonal constraints is to remove first (or on-the-fly) diagonal constraints as we know they can be removed (see [BDGP98]). However this leads to an unavoidable exponential blowup in the size of the model, as stated by the theorem below, hence in the length of the forward iterative computation.

☞ **Theorem 3.8 ([BC05])** *Timed automata with diagonal constraints are exponentially more concise*[8] *than timed automata without diagonal constraints.*

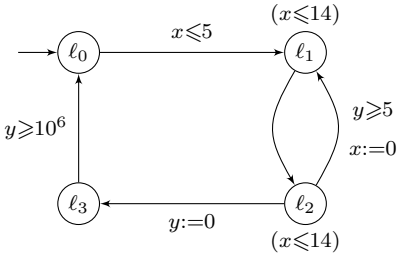
[8]This means that we can find a family of timed automata with diagonal constraints $(\mathcal{A}_n)_{n\in\mathbb{N}}$ such that the size of $\mathcal{A}_n$ is polynomial in n, and for every $n \in \mathbb{N}$, any timed automaton $\mathcal{B}_n$ without diagonal constraints that recognizes the same language as $\mathcal{A}_n$ will have size at least exponential in n.

In [BLR05b], together with François Laroussinie and Pierre-Alain Reynier (LSV/CNRS & ENS Cachan, France), we have proposed an algorithm to analyse timed automata with diagonal constraints without first removing diagonal constraints in the model. This algorithm is based on the abstraction-refinement paradigm [Kur94], and first applies the abstract forward analysis algorithm (that is correct for diagonal-free timed automata), and in case a path is found, that witnesses the reachability property, we check whether this is a real witness for the reachability property. In case it is, we have proven that the reachability property was satisfied. In case it is a spurious witness (*i.e.*, no real run can be read on that path), we look for the diagonal constraint that is 'responsible' for this incorrect answer (there must be one, thanks to Theorem 3.5), we remove it, and we start again the computation on the modified automaton. It is worth noticing that this diagonal constraint is non-trivial to be found, as there might be several (sometimes counter-intuitive) reasons for a path to be spurious, see [BLR05b] for details.

3.6.7 Improving the extrapolation operator

In this subsection, we focus on diagonal-free timed automata. We have seen in Theorem 3.5 that the abstraction operator Extra_k where k is the maximal constant appearing in the timed automaton can actually be improved. We will briefly present two ideas for improving the abstract forward computation. They are orthogonal, always improve the basic extrapolation operator, and can be applied together.

A location-dependent abstraction operator. The first improvement comes from the observation that the use of an extrapolation constant might not really be relevant in all the parts of an automaton. For instance, consider the timed automaton depicted below. The basic abstraction operator considers that the maximal constant (for clock y, if — following remark 3.6 — we consider one constant per clock) used in the extrapolation operator be 10^6. However, in location ℓ_1, from all configurations (ℓ_1, v) and (ℓ_1, v') with $v(x) = v'(x)$ and $v(y), v'(y) > 5$, the set of reachable states is identical (because the clock y is reset before its value is checked larger than 10^6), somehow stipulating the irrelevance of constant 10^6 in location ℓ_1.



We have developed a method that computes one extrapolation constant per pair '(clock,location)'. The computation of such constants is easy, and reduces to computing (least) solutions of difference bound inequations.

Each time there is a constraint that compares a clock x to a constant c on a transition leaving location ℓ, we add a constraint $k_x^\ell \geqslant c$ to the system of inequations, because the precise value of clock x up to c is useful to know whether the transition can be taken or not. Each time there is a transition that does not reset clock x between locations ℓ and ℓ', we add a constraint $k_x^\ell \geqslant k_x^{\ell'}$, expressing that if a constant has to be distinguished from location ℓ', then it also has to be distinguished from ℓ, because clock x is not reset. We do not add such a constraint if clock x is reset (as this somehow resets the dependence). To any timed automaton $\mathcal{A}$ we associate that way a system of inequations over variables $(k_x^\ell)_{(x,\ell)\in X\times L}$, that we denote $\mathcal{S}_\mathcal{A}$. Taking a tuple $K = (\alpha_x^\ell)_{(x,\ell)\in X\times L}$ of integers, which is a solution to the above system, we define in a natural way the extrapolation operator Extra_K that takes those constants as references (the choice of the extrapolation constant hence depends on the current location).

☞ **Theorem 3.9 ([BBFL03])** *Let $\mathcal{A}$ be a diagonal-free timed automaton. Consider the system of inequations $\mathcal{S}_\mathcal{A}$ as defined above, and consider K a (or the least) solution to this system. Then Extra_K is a correct abstraction with respect to reachability properties in $\mathcal{A}$.*

***Example* 3.10** Consider again the previous automaton. The system of inequations that corresponds to that automaton is:

$$\left\{\begin{array}{cccc} k_x^{\ell_0} \geqslant 5, k_x^{\ell_1} & k_x^{\ell_1} \geqslant 14, k_x^{\ell_2} & k_x^{\ell_2} \geqslant 14, k_x^{\ell_3} & k_x^{\ell_3} \geqslant k_x^{\ell_0} \\ k_y^{\ell_0} \geqslant k_y^{\ell_1} & k_y^{\ell_1} \geqslant k_y^{\ell_2} & k_y^{\ell_2} \geqslant 5 & k_y^{\ell_3} \geqslant 10^6, k_y^{\ell_0} \end{array}\right\}$$

The least solution to that system is $K = (\alpha_x^{\ell_i})$ with $\alpha_x^{\ell_i} = 14$ for every $i \in \{0,1,2,3\}$, $\alpha_y^{\ell_i} = 5$ for every $i \in \{0,1,2\}$, and $\alpha_y^{\ell_3} = 10^6$. The abstract forward analysis computation will be much shorter if we use the abstraction operator Extra_K than if we use the abstraction operator Extra_{10^6}. ⌟

***Remark* 3.11** Note that the system $\mathcal{S}_{\mathcal{A}}$ always has a solution, for instance the one that associates to every pair (x, ℓ) the maximal constant that appears in the timed automaton. Moreover, due to the form of the system of inequations, it also always has a (unique) least solution.

Furthermore notice that this method has originally been developed for the larger class of *updatable timed automata* [BDFP04]. For this general undecidable class of models, the existence of a solution to the system $\mathcal{S}_{\mathcal{A}}$ yields a correct algorithm for checking reachability properties, and moreover, for any decidable subclass described in [BDFP04], there is a solution to the system $\mathcal{S}_{\mathcal{A}}$. ⌟

Distinguishing between clock constraints. We first explain the basic idea of this new abstraction on a very simple example. Consider a zone over a single clock x defined by the constraint $a \leqslant x \leqslant b$ where a and b are integers. To detect whether an *upper-bounded* constraint $x \leqslant c$ can be satisfied, it is sufficient to know that $a \leqslant c$. The position of b with respect to c is not relevant. Hence, if only upper-bounded constraints are used, the idea is to abstract away this value, and to abstract the previous zone into the one defined by the constraint $a \leqslant x$. Similarly, to detect whether a *lower-bounded* constraint $x \geqslant c$ can be satisfied, it is sufficient to know that $c \leqslant b$. The position of a with respect to c is not relevant. Hence, if only lower-bounded constraints are used, the idea is to abstract away this value, and to abstract the previous zone into the one defined by the constraint $x \leqslant b$.

This idea can be formalized and generalized as follows: if $\mathcal{A}$ is a timed automaton, to every clock x of $\mathcal{A}$, we associate two constants $L(x)$ and $U(x)$ which respectively represent the maximal lower and upper bounds to which x is compared to in $\mathcal{A}$. They are called the *lower and upper bound functions* of $\mathcal{A}$. Then, if $M = (\prec_{i,j}; m_{i,j})_{0 \leqslant i,j \leqslant n}$ is a DBM in normal form, we define the DBM $\mathsf{Extra}_{LU}(M) = (\prec'_{i,j}; m'_{i,j})$ as follows:

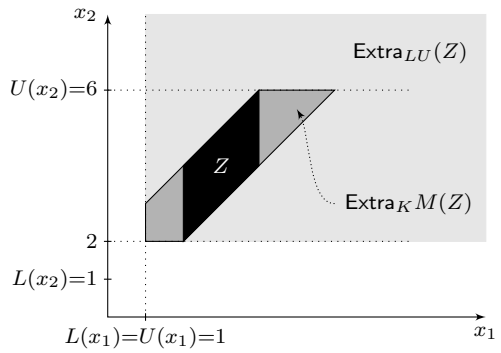
$$(\prec'_{i,j}; m'_{i,j}) = \begin{cases} (<; \infty) & \text{if either } m_{i,j} \geqslant L(x_i), \text{ or } -m_{0,i} \geqslant L(x_i), \\ & \quad \text{or } (i > 0 \text{ and } -m_{0,j} \geqslant U(x_j)) \\ (<; -U(x_j)) & \text{if } i = 0 \text{ and } -m_{0,j} \geqslant U(x_j) \\ m_{i,j} & \text{otherwise} \end{cases}$$

It is rather easy to get convinced that for every zone Z, it holds that $Z \subseteq \mathsf{Extra}_K(Z) \subseteq \mathsf{Extra}_{LU}(Z)$ if $K(x) = \max(L(x), U(x))$ for every clock $x \in X$. Hence if we use the abstraction operator Extra_{LU} instead of Extra_K in the forward analysis computation, it will always terminate faster.

***Example* 3.12** We consider the zone Z defined by the constraints

$$2 \leqslant x_1 \leqslant 4 \quad \wedge \quad 0 \leqslant x_2 - x_1 \leqslant 2$$

We assume $L(x_1) = U(x_1) = K(x_1) = 1$, and that $L(x_2) = 1$, $U(x_2) = K(x_2) = 6$. The abstracted zones $\mathsf{Extra}_K(Z)$ and $\mathsf{Extra}_{LU}(Z)$ are depicted below.



⌟

☞ **Theorem 3.13 ([BBLP04, BBLP05])** *Let $\mathcal{A}$ be a diagonal-free timed automaton. Define the lower and upper bound functions L and U of $\mathcal{A}$. Then Extra_{LU} is a correct abstraction with respect to reachability properties in $\mathcal{A}$.*

This is worth noticing that this abstraction is correct with respect to reachability properties, but that we may be very careful if one wants to use it for checking other kinds of properties. Indeed, the correctness proof of the above theorem relies on a notion of time-abstract simulation (which does not preserve deadlock properties), whereas the proof of *eg.* Theorem 3.5 relied on a stronger notion of time-abstract bisimulation.

***Remark* 3.14** Note that the inclusion check used for stopping earlier the forward computation together with the use of the abstraction operator Extra_{LU} encompasses the domination point trick used for improving the analysis of jobshop scheduling problems [AM01, AAM06].

⌟

Practical improvements. These two coarser abstraction operators have been implemented in the tool Uppaal (since the version 3.4.2 of the tool). Improvements in the performance have been rather impressive, with a speed-up of approximately 20% of the analysis times. We do not report the experiments here but better refer to [BBLP05].

3.7 Conclusion

In this chapter, we have rather briefly presented works done for improving the verification of basic properties (like reachability properties) in timed automata. These works have mostly focused on the development of algorithms

and abstractions for the analysis of timed automata. The basic algorithms have been presented, together with a commonly used symbolic representation. We have also reported some abstractions which have yielded much improvement in the practical verification of timed systems (they are now standard options in the tool Uppaal). This work stems from collaborations started during my post-doctoral stay at Aalborg Universitet (Denmark) in 2002. The works on the abstraction operators have been done in collaboration with Gerd Behrmann, Emmanuel Fleury, Kim G. Larsen, and Radek Pelànek.

Since then, I am not aware of the development of any other abstraction operators that have improved that much the analysis of timed automata. In that area, works have mostly focused on the development of better data structures for representing zones, and on algorithmic ideas to improve the basic operations on zones (see for instance the work on federations made by the Uppaal team [DHGP04, Dav05]).

Chapter 4

Linear-time temporal logics for real-time systems

4.1 Introduction

In the previous chapter, we have seen techniques developed for verifying basic properties (like reachability, safety, *etc.*) in timed automata. These techniques extend to untimed specification languages like LTL [Pnu77], but do not straightforwardly extend to more involved properties that express timing constraints. For instance, one would like to express bounded-response time properties like 'the airbag inflates within 5 ms after the car crashes', which imposes not only a constraint on the sequence of events that happen, but also on the delays between those events. Timed automata are adequate to represent timed systems, but not that much for representing properties of systems. Indeed, if $\mathcal{A}$ is a timed automaton representing the system, and $\mathcal{P}$ a timed automaton representing the property, verifying that $\mathcal{A}$ satisfies the property $\mathcal{P}$ corresponds to checking that all behaviours of $\mathcal{A}$ are also behaviours of $\mathcal{P}$. This is an inclusion question, and that problem is unfortunately undecidable for timed automata [AD94]. Of course, one can then model undesired behaviours (*i.e.* the negation of the property we want to verify) in a timed automaton $\mathcal{P}$, in which case we now need to check that there is no joint behaviours in $\mathcal{A}$ and $\mathcal{P}$ (like in test automata, see [ABBL03]), but this is actually not possible to express that way properties like response properties.

Following the development of temporal logics in model-checking of finite-state systems [Pnu77, QS82, SC85, CES86] (*cf.* section 2.2.2), timed temporal logics have been proposed, which extend classical temporal logics with timing constraints. There are several ways of expressing such constraints, a

standard one consists in constraining temporal modalities. For instance, one can write a formula like

$$\mathbf{G}\,(\mathsf{car.crash} \rightarrow \mathbf{F}_{\leqslant 5\,\mathrm{ms}}\,\mathsf{airbag.inflate})$$

to express the above-mentioned *quantitative* property of the airbag. Several timed extensions of CTL [CES86] and LTL [Pnu77] have been proposed, and TCTL, a natural extension of CTL, has been first proven suitable (from a decidability point-of-view) for model-checking purposes [ACD90, ACD93, HNSY94]. On the contrary, model-checking natural timed extensions of LTL, like MTL [Koy90] or TPTL [AH89], is very hard [AH93, Hen98, OW05].

In this chapter our point is not to discuss the advantages or disadvantages of the branching-time *vs* the linear-time paradigms (for that we refer to *eg.* [Var98, Var01, NV07]), nor it is to give an exhaustive overview of all the works on timed temporal logics. It is rather to focus on the linear-time framework, which has generated quite a wide range of works these last few years in the timed systems community, and to give a comprehensive overview of the expressiveness and decidability of main linear-time timed temporal logics.

After having presented several logics extending LTL with timing constraints in section 4.2, we will present a first expressiveness result that had been conjectured in the early nineties, and that we have proven recently (section 4.3). Then, we will turn to the decidability of those logics (for the classical model-checking and satisfiability problems), and show that it is seldom decidable and always difficult! We will explain why this is actually the case, by informally showing how we can simulate channel machines using linear-time timed temporal logics (section 4.4). We will end this chapter (section 4.5) with a rather long section on fragments of logics that have been studied, yielding much more interesting decidability and complexity results.

4.2 Syntax and semantics of linear-time timed temporal logics

In this section, we present the logics we will focus on in the whole chapter. All these logics extend the logic LTL [Pnu77] by adding timing constraints in the formulas. In the literature, several ways of doing so have been proposed, yielding logics with different properties and expressiveness, we can mentioned for instance MTL [Koy90], TPTL [AH89], QTL [HR04, HR05], or TL+counting [HR07, Rab08]. In this chapter we focus on the two first logics, as they are the ones we have worked on these last few years.

We let AP be a finite set of atomic propositions.

4.2.1 The logic MTL

The syntax of MTL [Koy90] over AP is given by the following grammar:

$$\mathsf{MTL} \ni \varphi \ ::= \ p \ \mid \ \neg\varphi \ \mid \ \varphi \vee \varphi \ \mid \ \varphi \, \mathbf{U}_I \, \varphi$$

where $p \in \mathsf{AP}$, and I is an interval of $\mathbb{R}_{\geqslant 0}$ with rational bounds.

In chapter 2, we have defined two semantics for LTL. We make the same distinction of MTL, and distinguish between the pointwise and the continuous semantics. Hence, following what we have done for LTL, we give the semantics of MTL with a generic notion of *position*. We refer to page 19 for the two possible interpretations of this term.

Formulas of MTL are interpreted over runs of a timed automaton, from some position along that run.[1] Basic formulas are interpreted as for LTL, we thus omit the definitions. We fix a timed automaton $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L})$, and we let $\varrho = s \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \cdots s_{n-1} \xrightarrow{t_n,e_n} s_n \cdots$ be a finite or infinite run of $\mathcal{A}$, and ϖ be a position along ϱ. The satisfaction relation for the Until modality is defined as follows:

$$\begin{array}{rcl} (\varrho, \varpi) \models \varphi \, \mathbf{U}_I \, \psi & \Leftrightarrow & \text{there exists a position } \varpi' > \varpi \text{ along } \varrho \text{ such that} \\ & & \mathsf{duration}(\varrho_{[\varpi;\varpi']}) \in I, \ (\varrho, \varpi') \models \psi, \\ & & \text{and for every position } \varpi < \varpi'' < \varpi', \ (\varrho, \varpi'') \models \varphi \end{array}$$

where $\varrho_{[\varpi;\varpi']}$ is the sub-run of ϱ that starts at position ϖ and ends at position ϖ', and $\mathsf{duration}(\cdot)$ gives the duration of a run, *i.e.*, the sum of all delays along that run. If ϖ_0 is the initial position of run ϱ, we write $\varrho \models \varphi$ whenever $(\varrho, \varpi_0) \models \varphi$. Let $\mathcal{A}$ be a timed automaton with initial configuration s_0, and φ be an MTL formula. We say that $\mathcal{A}$ satisfies φ over infinite (respectively finite) runs, and we write $\mathcal{A} \models \varphi$ (respectively $\mathcal{A} \models_{\mathsf{f}} \varphi$), whenever for all infinite runs $\varrho \in \mathsf{Runs}(\mathcal{A}, s_0)$ (respectively $\varrho \in \mathsf{Runs}^{\mathsf{acc}}_{\mathsf{f}}(\mathcal{A}, s_0)$), $\varrho \models \varphi$.

As we will need to really distinguish between the semantics in this chapter, we introduce some more notations. When it will be required, we will write $\models^{\mathsf{cont}}$ (respectively $\models^{\mathsf{point}}$) for the satisfaction relation under the continuous (respectively pointwise) semantics. Hence, in the following we may distinguish between the four satisfaction relations $\models^{\mathsf{cont}}_{\mathsf{f}}$, $\models^{\mathsf{point}}_{\mathsf{f}}$, $\models^{\mathsf{cont}}$, and $\models^{\mathsf{point}}$.

[1] Usually, MTL is interpreted over timed words (or equivalently observations) [Hen98, Ras99] or over signals (or equivalently timed state sequences) [AH93, AH94, AFH96, Ras99], but this formulation with runs is simpler and sufficient for our purpose.

We extend the syntactic sugar defined in subsection 2.2.2 for LTL: $\mathbf{F}_I\,\varphi \equiv (\mathtt{true}\,\mathbf{U}_I\,\varphi)$ (eventually, φ will hold within interval I from now), $\mathbf{G}_I\,\varphi \equiv \neg(\mathbf{F}_I\,\neg\varphi)$ (for all positions within I, φ holds), and $\mathbf{X}_I\varphi \equiv (\mathtt{false}\,\mathbf{U}_I\,\varphi)$ (next position is within I from now and satisfies φ). We also use pseudo-arithmetical expressions to represent intervals. For instance, '$=1$' stands for the singleton interval $[1;1]$, and '$\geqslant 2$' stands for the interval $[2;+\infty)$.

***Example* 4.1** Using MTL, we can write properties like

$$\mathbf{G}\,(\mathsf{problem} \rightarrow \mathbf{F}_{\leqslant 2}\,\mathsf{alarm}) \tag{4.1}$$

expressing that each time a problem occurs, within 2 time units, an alarm rings.

We can also express more involved properties, like

$$\mathbf{G}\,(\mathsf{problem} \rightarrow (\mathbf{F}_{\leqslant 15}\,\mathsf{repair} \vee \mathbf{G}_{[12,15]}\,\mathsf{alarm}))$$

saying that each time a problem occurs, either it is repaired in no more than 15 time units, or an alarm rings for 3 time units 12 time units after the problem. ⌟

***Remark* 4.2** The choice of the interpretation of MTL in terms of the pointwise or the continuous semantics has an impact on the precise meaning of the formulas, and as we will see later, also on their applicability in model-checking. The formula $\mathbf{F}_{=2}\,a$ expresses that an a will happen two time units later. This formula is equivalent to $\mathbf{F}_{=1}\,\mathbf{F}_{=1}\,a$ (in one time unit, it will be the case that in one time unit, an a occurs) in the continuous semantics, but not in the pointwise semantics, because there may be no action one time unit after the initial position (hence any formula $\mathbf{F}_{=1}\,\psi$ would be evaluated as wrong from the initial configuration). ⌟

4.2.2 Two extensions of MTL: TPTL and MTL+Past

The logic TPTL. In MTL, timing constraints are added using intervals decorating modalities. There is another classical way for expressing such quantitative constraints, which consists in adding variables to the formulas. The logic TPTL [AH89] over AP and the (finite) set of variables Y is defined by the following grammar:

$$\mathsf{TPTL} \ni \varphi \;::=\; p \;\mid\; \neg\varphi \;\mid\; \varphi \vee \varphi \;\mid\; x.\varphi \;\mid\; x \in I \;\mid\; \varphi\,\mathbf{U}\,\varphi$$

where $p \in \mathsf{AP}$, I is an interval of $\mathbb{R}_{\geqslant 0}$ with rational bounds, and $x \in Y$ is a formula variable.

The semantics of TPTL is defined on finite or infinite runs of a timed automaton with a valuation for formula variables, and a position along the run. We fix a timed automaton $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L})$, and we let $\varrho = s \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \cdots s_{n-1} \xrightarrow{t_n,e_n} s_n \cdots$ be a finite or infinite run,

$u \in \mathbb{R}^Y_{\geqslant 0}$ be a valuation for formula variables, and ϖ be a position along ϱ. The satisfaction relation is then defined as follows:

$$\begin{array}{rcl}
(\varrho, \varpi, u) \models p & \Leftrightarrow & p \in \mathcal{L}(\varrho_{[\varpi]}) \\
(\varrho, \varpi, u) \models \neg\varphi & \Leftrightarrow & (\varrho, \varpi, u) \not\models \varphi \\
(\varrho, \varpi, u) \models \varphi \vee \psi & \Leftrightarrow & (\varrho, \varpi, u) \models \varphi \text{ or } (\varrho, \varpi, u) \models \psi \\
(\varrho, \varpi, u) \models x.\varphi & \Leftrightarrow & (\varrho, \varpi, u[x \leftarrow \mathsf{time}(\varpi)]) \models \varphi \\
(\varrho, \varpi, u) \models x \in I & \Leftrightarrow & \mathsf{time}(\varpi) - u(x) \in I \\
(\varrho, \varpi, u) \models \varphi\, \mathbf{U}_I\, \psi & \Leftrightarrow & \text{there exists a position } \varpi' > \varpi \text{ along } \varrho \text{ such that} \\
 & & \mathsf{duration}(\varrho_{[\varpi;\varpi']}) \in I,\ (\varrho, \varpi', u) \models \psi, \\
 & & \text{and for every position } \varpi < \varpi'' < \varpi',\ (\varrho, \varpi'', u) \models \varphi
\end{array}$$

where $\mathsf{time}(\varpi) \stackrel{\text{def}}{=} \mathsf{duration}(\varrho_{[\varpi_0;\varpi]})$ (ϖ_0 is the initial position along ϱ), and $u[x \leftarrow \alpha]$ is the valuation assigning $u(y)$ to every variable $y \in Y \setminus \{x\}$, and α to x.

We use the two interpretations for the term position, as in MTL, and when useful, we distinguish between the two satisfaction relations $\models^{\mathsf{cont}}$ and $\models^{\mathsf{point}}$. As for MTL, we define the satisfaction relation for a timed automaton over infinite or finite runs. Also we use the same syntactic sugar as for MTL ($\mathbf{F}$, $\mathbf{G}$, *etc.*).

***Example* 4.3** This is not difficult to get convinced that property (4.1) can be rewritten in TPTL as the formula

$$\mathbf{G}\,(\mathsf{problem} \rightarrow x.\mathbf{F}\,(\mathsf{alarm} \wedge x \leqslant 2))$$

Indeed, each time a problem occurs, the current value of the time is frozen and stored in the formula variable x, and then later, when the alarm rings, we verify that the delay since the value has been frozen is smaller than or equal to 2.

Another TPTL formula is:

$$\mathbf{G}\,(\mathsf{problem} \rightarrow x.\mathbf{F}\,(\mathsf{alarm} \wedge \mathbf{F}\,(\mathsf{failsafe} \wedge x \leqslant 2))) \tag{4.2}$$

which says that whenever a problem occurs, then within 2 time units, an alarm rings and later (but still within 2 time units since the problem occurred), the system enters a failsafe mode.

⌟

The logic MTL+Past. Following the classical untimed framework [Kam68, LPZ85], we also extend MTL with past-time modalities. The syntax of MTL+Past [AH92a, AH93] over AP is given by the following grammar:

$$\mathsf{MTL{+}Past} \ni \varphi \ ::= \ p \ \mid \ \neg\varphi \ \mid \ \varphi \vee \varphi \ \mid \ \varphi\, \mathbf{U}_I\, \varphi \ \mid \ \varphi\, \mathbf{S}_I\, \varphi$$

where $p \in \mathsf{AP}$, and I is an interval of $\mathbb{R}_{\geqslant 0}$ with rational bounds. The $\mathbf{S}$-modality is called the 'Since' modality, and is somehow the dual of the 'Until'.

The semantics of all modalities except the 'Since' have already been given. We fix a timed automaton $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L})$, we let $\varrho = s \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \cdots s_{n-1} \xrightarrow{t_n,e_n} s_n \cdots$ be a finite or infinite run of $\mathcal{A}$, and ϖ be a position along ϱ. The satisfaction relation is defined as follows:

$$(\varrho, \varpi) \models \varphi \,\mathbf{S}_I\, \psi \;\Leftrightarrow\; \begin{array}{l}\text{there exists a position } \varpi' < \varpi \text{ along } \varrho \text{ such that} \\ \mathsf{duration}(\varrho_{[\varpi';\varpi]}) \in I,\ (\varrho, \varpi') \models \psi, \\ \text{and for every position } \varpi' < \varpi'' < \varpi,\ (\varrho, \varpi'') \models \varphi\end{array}$$

The intuition of formula $\varphi \,\mathbf{S}_I\, \psi$ is that φ holds since ψ was true (within I in the past). In a standard way, we define $\mathbf{F}_I^{-1}\, \varphi \equiv (\mathtt{true}\, \mathbf{S}_I\, \varphi)$, which says that φ was true in the past, within a delay belonging to the interval I.

***Example* 4.4** The formula

$$\mathbf{G}\,(p \rightarrow \mathbf{F}_{=1}^{-1}\, q)$$

expresses that every p is preceded one time unit earlier by a q. ⌟

4.3 Expressiveness of linear-time timed temporal logics

If needed, some vocabulary is defined in Table 4.1. In that section, we focus on infinite runs.

In a rather obvious way, we get that every MTL formula can be expressed in MTL+Past (syntactical inclusion), and in TPTL. Indeed, to translate an MTL formula into an equivalent TPTL formula, it is sufficient to replace any $\varphi \,\mathbf{U}_I\, \psi$-sub-formula by $x.(\varphi \,\mathbf{U}\, (\psi \wedge x \in I))$ where x is a fresh variable dedicated to that sub-formula. It is not difficult to check that this transformation preserves the equivalence of formulas.

It had been conjectured for awhile (see [AH90, AH92b, AH93, Hen98]) that TPTL is strictly more expressive than MTL, with the suggestion that formula (4.2) should be a witness of that gap of expressiveness. Together with Fabrice Chevalier and Nicolas Markey, we have worked on that conjecture and have proven the following surprising result, partly contradicting the fact that formula (4.2) is a witness of the conjecture.

Let φ and φ' be two formulas that are interpreted over timed automata. They are said *equivalent* whenever for every timed automaton $\mathcal{A}$, $\mathcal{A}$ satisfies φ if and only if $\mathcal{A}$ satisfies φ'. Let $\mathcal{L}$ and $\mathcal{L}'$ be two logical languages interpreted over timed automata. We say that $\mathcal{L}'$ is *at least as expressive as* $\mathcal{L}$, and we write $\mathcal{L} \leqslant \mathcal{L}'$, whenever for every formula $\varphi \in \mathcal{L}$, there is a formula $\varphi' \in \mathcal{L}'$ which is equivalent to φ. We say that $\mathcal{L}'$ is *strictly more expressive* than $\mathcal{L}$ whenever $\mathcal{L} \leqslant \mathcal{L}'$ and there exists a formula $\varphi' \in \mathcal{L}'$ such that for every formula $\varphi \in \mathcal{L}$, φ and φ' are not equivalent (we then say that φ' cannot be expressed in $\mathcal{L}$). Finally we say that $\mathcal{L}$ and $\mathcal{L}'$ are *equally expressive* whenever $\mathcal{L} \leqslant \mathcal{L}'$ and $\mathcal{L}' \leqslant \mathcal{L}$.

There are several methods to prove that a logic $\mathcal{L}'$ is strictly more expressive than a logic $\mathcal{L}$. First one can prove that there are two systems, say two timed automata $\mathcal{A}$ and $\mathcal{B}$ in our case, and a formula φ' in $\mathcal{L}'$ such that $\mathcal{A} \models \varphi'$, $\mathcal{B} \not\models \varphi'$, but for every formula φ in $\mathcal{L}$, $\mathcal{A} \models \varphi$ if and only if $\mathcal{B} \models \varphi$. We then speak of the *distinguishing power* of the logics. Sometimes, it may be the case that $\mathcal{L}'$ is strictly more expressive than $\mathcal{L}$, but that they have the same distinguishing power (this is for instance the case of LTL, and of the logic that only allows the $\mathbf{X}$-modality). In all the cases we will consider in this chapter, this will actually be the case, and we will need more involved constructions.

An example of more involved method is as follows. We construct a formula φ' in $\mathcal{L}'$ and two (infinite) families of timed automata $(\mathcal{A}_i)_{i\in\mathbb{N}}$ and $(\mathcal{B}_i)_{i\in\mathbb{N}}$ such that for every $i \in \mathbb{N}$, $\mathcal{A}_i \models \varphi'$, $\mathcal{B}_i \not\models \varphi'$, and for every formula φ in $\mathcal{L}$, there exists $i \in \mathbb{N}$ such that $\mathcal{A}_i \models \varphi$ if and only if $\mathcal{B}_i \models \varphi$.

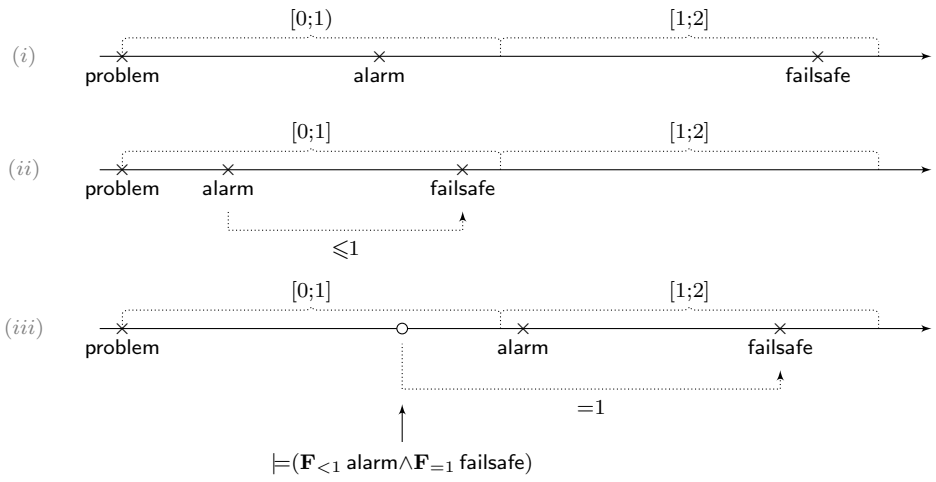
Later (see table 4.2), we will give an example.

Table 4.1: About the expressiveness of logical languages

☞ **Proposition 4.5 ([BCM05])** *In the* ***continuous semantics****, formula (4.2) is equivalent to the* MTL *formula*

$$\mathbf{G}\left(\mathsf{problem} \rightarrow \left(\begin{array}{ll} & \mathbf{F}_{<1}\,\mathsf{alarm} \wedge \mathbf{F}_{[1,2]}\,\mathsf{failsafe} \\ \vee & \mathbf{F}_{\leqslant 1}\,(\mathsf{alarm} \wedge \mathbf{F}_{\leqslant 1}\,\mathsf{failsafe}) \\ \vee & \mathbf{F}_{\leqslant 1}\,(\mathbf{F}_{<1}\,\mathsf{alarm} \wedge \mathbf{F}_{=1}\,\mathsf{failsafe}) \end{array}\right)\right) \begin{array}{l} \leftarrow (i) \\ \leftarrow (ii) \\ \leftarrow (iii) \end{array}$$

We explain the three sub-cases appearing in the formula of proposition 4.5, illustrated on the picture next page. Cases (i) and (ii) are rather easy to understand: on the first figure, the alarm happens within 1 time unit after problem, and failsafe at least 1 time unit, but no more than 2 time units, after alarm (which can be expressed by the MTL formula $\mathbf{F}_{<1}\,\mathsf{alarm} \wedge \mathbf{F}_{[1;2]}\,\mathsf{failsafe}$); on the second figure, alarm and failsafe happen within 1 time unit after alarm (this can be 'over-approximated' by the formula $\mathbf{F}_{\leqslant 1}\,(\mathsf{alarm} \wedge \mathbf{F}_{\leqslant 1}\,\mathsf{failsafe})$).



Case (*iii*) is more tricky: the idea is to say that there is some (virtual point) within 1 time unit after problem that will be exactly 1 time units before failsafe, and it remains to say that alarm then needs to be within the interval $(0;1)$ after this virtual point. This is expressed by the formula $\mathbf{F}_{\leqslant 1}\,(\mathbf{F}_{<1}\,\mathsf{alarm} \wedge \mathbf{F}_{=1}\,\mathsf{failsafe})$. As there may be no action at the virtual point, this formula does not express what we want in the pointwise semantics, hence it is correct (or does correspond to our intuition) **only in the continuous semantics**!

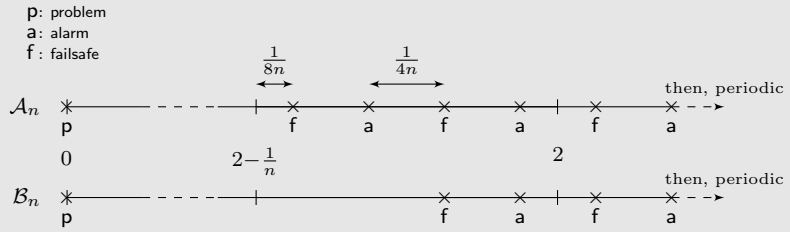
We have actually proven that formula (4.2) cannot be expressed in MTL under the pointwise semantics (see table 4.2 for two families of automata that distinguish between MTL and formula (4.2)). This proves the conjecture that TPTL is strictly more expressive than MTL in the pointwise semantics. To prove the conjecture in the continuous semantics, we have exhibited another formula, which says that there is a p within the first time unit, so that for the rest of the first time unit, q does not hold: $x.\mathbf{F}\,(p \wedge x \leqslant 1 \wedge \mathbf{G}\,(x \leqslant 1 \rightarrow \neg q))$. We have proven that this formula cannot be expressed in MTL in the continuous semantics, confirming the initial conjecture.

☞ **Theorem 4.6 ([BCM05])** *The logic* TPTL *is strictly more expressive than* MTL*, under both pointwise and continuous semantics.*

We get similar results for MTL+Past, which requires exhibiting other formulas: $\mathbf{F}_{\leqslant 2}\,(q \wedge \mathbf{F}^{-1}\,p)$ in the pointwise semantics, and $\mathbf{F}_{=1}\,(\neg q\,\mathbf{S}\,p)$ in the continuous semantics. We thus also get the following theorem.

☞ **Theorem 4.7 ([BCM05])** *The logic* MTL+Past *is strictly more expressive than* MTL*, under both pointwise and continuous semantics.*

We consider the two families of timed automata $(\mathcal{A}_n)_{n\in\mathbb{N}}$ and $(\mathcal{B}_n)_{n\in\mathbb{N}}$ that generate the two infinite executions depicted below.



We can check that for every $n \in \mathbb{N}$,

$$\begin{cases} \mathcal{A}_n \models^{\mathsf{point}} \mathbf{G}\left(\mathsf{p} \to x.\Big(\mathbf{F}\,(\mathsf{a} \wedge \mathbf{F}\,(\mathsf{f} \wedge x \leqslant 2))\Big)\right) \\ \mathcal{B}_n \not\models^{\mathsf{point}} \mathbf{G}\left(\mathsf{p} \to x.\Big(\mathbf{F}\,(\mathsf{a} \wedge \mathbf{F}\,(\mathsf{f} \wedge x \leqslant 2))\Big)\right) \end{cases}$$

On the other hand, we can prove that for every $n \in \mathbb{N}$, for every formula $\varphi \in \mathsf{MTL}$ of granularity $\frac{1}{n}$,[a]

$$\mathcal{A}_n \models^{\mathsf{point}} \varphi \quad \Leftrightarrow \quad \mathcal{B}_n \models^{\mathsf{point}} \varphi$$

For some of the other expressiveness results that we have proven in [BCM05], more complex models have been required, with two parameters, one for the granularity and the other one for the size of the formula.

[a] All constants appearing in the formula are integral multiples of $\frac{1}{n}$.

Table 4.2: Formula (4.2) cannot be expressed in MTL in the pointwise semantics

In [BCM05] we have proven some more expressiveness that we will not detail here, for instance that MITL [AFH96], the fragment of MTL where intervals labelling an $\mathbf{U}$-modality cannot be punctual, is strictly less expressive than MTL in both semantics. Various expressiveness results pursuing our work and comparing MTL+Past, MTL, TPTL in their different semantics can be found in the recent literature [DP06, DP07, DRP07].

4.4 The model-checking and satisfiability problems

Definition of the problems. We let $\mathcal{L}$ be a logical language interpreted over (runs of) timed automata. The *model-checking problem* for the class $\mathcal{L}$ asks, given a formula $\varphi \in \mathcal{L}$ and a timed automaton $\mathcal{A}$, whether $\mathcal{A} \models \varphi$.

We let $\mathcal{U}$ be the universal timed automaton, *i.e.* the timed automaton that reads and accepts everything. The *satisfiability problem* for $\mathcal{L}$ asks, given a formula $\varphi \in \mathcal{L}$, whether there is a run ϱ in $\mathcal{U}$ such that $\varrho \models \varphi$. This is equivalent to '$\mathcal{U} \not\models \neg\varphi$'. This definition is rather non-standard where it is better defined as the existence of a timed word (or time-state sequence) satisfying the property φ, but this is due to our choice of the simplified semantics of the logics on runs of timed automata.

In the two cases, we of course distinguish variants of the problems, depending on the choice of the semantics.

Discussion. Until very recently [OW05], MTL and TPTL were both considered as undecidable, as they are both able to express the so-called forward propagating formula '$\mathbf{G}\,(p \rightarrow \mathbf{F}_{=1}\,q)$', and the claim was that "as soon as a logic is powerful enough to express the forward propagating formula, it is undecidable" [AH92a, AH93, Hen98]. However that was a bit misleading, as the decidability subtly depends on the choice of the semantics (being either pointwise or continuous, and either interpreted over finite or infinite runs). Below we summarize the decidability results for the model-checking problem[2] for the logics MTL, MTL+Past and TPTL, indicating in the columns the satisfaction relation that is considered.

	$\models_{\mathsf{f}}^{\mathsf{point}}$	$\models_{\mathsf{f}}^{\mathsf{cont}}$	$\models^{\mathsf{point}}$	$\models^{\mathsf{cont}}$
MTL	decidable, NPR [OW05, OW07]	undecidable [AFH96]	undecidable [OW06a]	undecidable [AFH96]
MTL+Past	undecidable	undecidable	undecidable	undecidable
TPTL	undecidable [AH94]	undecidable [AH94]	undecidable [AH94]	undecidable [AH94]

Note that in the table above, any hardness result (*eg.* undecidability results) can easily be lifted from the pointwise to the continuous semantics.

[2]Note that these complexities coincide with those of the satisfiability problem, because the considered logics are closed by negation, and timed automata can be encoded in a polynomial size MTL — or more precisely an MITL — formula [HRS98, Ras99].

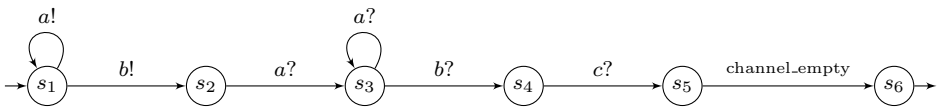
This shows that model-checking linear-time timed temporal logics over timed automata is really hard, the only logic that is decidable being MTL under the weakest interpretation (the $\models_{\mathsf{f}}^{\mathsf{point}}$-semantics). Moreover the complexity is very high, it is non-primitive recursive [Sch02, CS08]. In the next subsection, we explain some of the hardness results that we have mentioned above (those which concern finite runs). The way it is presented is inspired by [OW05], and has been worked out together with my PhD student Fabrice Chevalier [Che07] when we have worked on the control problem for MTL specifications [BBC06a]. In subsection 4.4.2, we explain the decidability of MTL for the $\models_{\mathsf{f}}^{\mathsf{point}}$-semantics (based on alternating timed automata), and explain why the techniques cannot be extended to the $\models^{\mathsf{point}}$-semantics. The only undecidability proof that we do not explain is the undecidability of MTL model-checking under the latter semantics [OW06a].

4.4.1 Model-checking linear-time timed properties is hard...

We first explain most of the lower bounds mentioned in the previous table. They are obtained *via* a reduction from the halting problem for channel machines with insertion errors and emptiness tests (ICMETs in short).

A *channel machine* [BZ83] is a finite automaton which can write on a channel and read from it following a FIFO policy. We note '$a!$' for writing a at the tail of the channel and '$a?$' for reading an a at the head of the channel. A channel machine has insertion errors if any letter can be written at any time anywhere in the channel. A channel machine without insertion errors is said perfect. The *halting problem* for a channel machine asks whether a distinguished halting state can be reached following rules of the machine. It is rather obvious that the halting problem for a channel machine with insertion error is trivial as any transition can be taken at any time (one can always write on a channel, and if one cannot read a letter, it is then possible to insert the relevant symbol on the channel). Hence, we add to the channel machine the capability of testing that the content of a channel is empty or not, yielding the ICMET model that we have already mentioned.

***Example* 4.8** Consider the channel machine depicted below:



A configuration of this system is a pair (s, w) where s is a discrete state of the machine and w is a word representing the content of the channel. We give an

error-free computation example for that machine, where ε represents the empty channel:

$$(s_1, \varepsilon) \xrightarrow{a!} (s_1, a) \xrightarrow{a!} (s_1, aa) \xrightarrow{b!} (s_2, aab) \xrightarrow{a?} (s_3, ab) \xrightarrow{a?} (s_3, b) \xrightarrow{b?} (s_4, \varepsilon)$$

We can see that no error-free computation allows to reach state s_6 (because no c is ever written on the channel). If we assume that this machine has insertion errors, then the following move is allowed:

$$(s_4, \varepsilon) \xrightarrow{c?} (s_5, \varepsilon)$$

(we assume implicitly that c has been inserted on the channel, so that the last transition labelled by 'c?' can now be fired). Then, s_6 is reachable because the channel is empty in configuration (s_5, ε), hence the last transition can be taken. ⌟

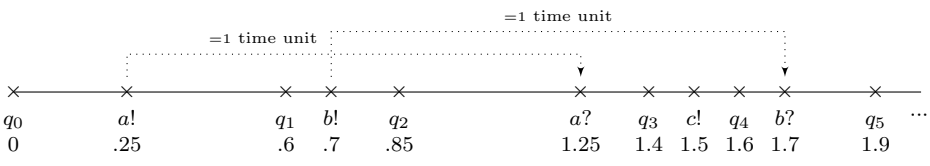
We recall results concerning the halting problem for channel machines.

Proposition 4.9

- *The halting problem for perfect channel machines is undecidable [BZ83].*
- *The halting problem for channel machines with insertion errors and emptiness tests is decidable [Fin94, AJ96, CFPI96] but non-primitive recursive [Sch02]. It is even in the $\mathfrak{F}_{\omega^\omega}$-level of the fast growing hierarchy [CS08].*[3]

We now explain how MTL (and variants thereof) can capture the behaviours of channel machines. We will build a formula φ so that φ is satisfiable if and only if the channel machine halts. Note that we will use the terminology 'timed word' in that part, to simplify the presentation (see page 19).

The idea is to encode a computation of a channel machine as a timed word. In this encoding, the underlying untimed word is the trace of the computation, that is, an alternating sequence of states and actions. We use timing constraints to enforce the channel be FIFO: we require that any write action 'a!' is followed one time unit later by a corresponding read action 'a?'. This is not difficult to be convinced that this enforces the channel be FIFO. We illustrate this encoding on the next figure, which represents a timed word (actions and time stamps).



[3]Formally, in those papers, that is the halting problem for lossy channel machines which is considered, but there is an easy reduction from the halting problem for lossy channel machines to the halting problem for ICMETs.

The above timed word encodes the following computation of the channel machine:

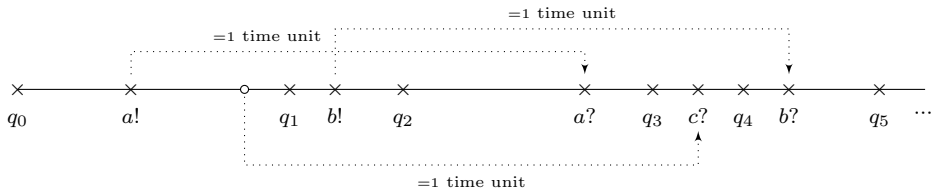
$$(q_0, \varepsilon) \xrightarrow{a!} (q_1, a) \xrightarrow{b!} (q_2, ab) \xrightarrow{a?} (q_3, b) \xrightarrow{c!} (q_4, bc) \xrightarrow{b?} (q_5, c) \cdots$$

To properly encode a behaviour of a channel machine, a timed word must satisfy the following constraints:

- states and actions alternate. This can be checked using an LTL formula.
- the untimed projection of the timed words follows the rules of the channel machine. This can also be encoded with an LTL formula.
- the emptiness test can be done by enforcing that the delay between the actions corresponding to the source and target states of the edge is 1 time unit. Due to the next rule, this will enforce the channel be empty.
- the channel is FIFO: to that aim we express that every write action is followed one time unit later by a corresponding read action. This can be expressed in MTL using formulas of the form:

  $$\mathbf{G}\ (a! \rightarrow \mathbf{F}_{=1}\, a?)$$

  However, this formula does not encode the property that the channel behaves properly. Indeed, nothing prevents a read event '$a?$' to happen, even though there is no corresponding write event '$a!$' one time unit earlier. For instance, consider the following timed word:



  This timed word satisfies the propagating formulas $\mathbf{G}\ (a! \rightarrow \mathbf{F}_{=1}\, a?)$ (for every letter a), even though the event '$c?$' is not preceded by any action one unit earlier. The above formula hence only encodes the behaviour of a channel machine *with* insertion errors. However, from that study, we already learn that the model-checking of MTL over finite runs in the pointwise semantics is non-primitive recursive. To encode a perfect channel machine, we need to be able to express the property that every '$a?$' is preceded one time unit earlier by an '$a!$'. We call this property the 'backward matching property'.

We now discuss how we can express the backward matching property in timed temporal logics. Indeed, we would like to know whether MTL can express or not the behaviour of a perfect channel machine. We will present here natural ideas, which will happen to be wrong for MTL, but sufficient to prove undecidability of several variants or extensions of MTL.

- A first simple idea is to express this 'backward matching property' using the following formula:

$$\mathbf{G}\ ((\mathbf{F}_{=1}\, a?) \to a!)$$

  which expresses the fact that if there is a read event '$a?$' one time unit later, then there must be right now a corresponding write event '$a!$'. It is not hard to see that in the pointwise semantics, this does not express what we want. Indeed this formula is still satisfied by the previous timed word, because there is no action one time unit before the action '$c?$'. However, in the continuous semantics, this formula really enforces a perfect behaviour of the FIFO channel. That is why MTL model-checking in the continuous semantics is undecidable.

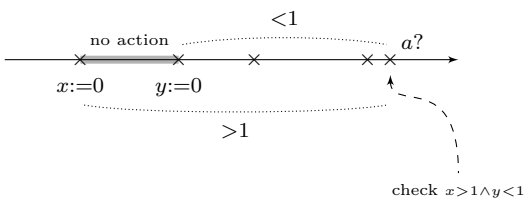
- A second idea is to express this 'backward matching property' using a past-time modality (hence in MTL+Past). The formula

$$\mathbf{G}\ \left(a? \to \mathbf{F}^{-1}_{=1}\, a!\right)$$

  precisely expresses that every read event '$a?$' is preceded one time unit earlier by a matching write event '$a!$'. That is why MTL+Past is undecidable, even in the pointwise semantics.

- Finally, the 'backward matching property' can be expressed in TPTL using the following more involved property:

$$\neg\Big(\mathbf{F}\, x \cdot \mathbf{X}\left(y \cdot \mathbf{F}\, (x > 1 \wedge y < 1 \wedge a?)\right)\Big)$$



  Informally (see the picture), this formula negates the fact that there are two consecutive positions (in the pointwise sense) such that an a

is read more than one time unit after the first position, and less than time unit after the second position. This precisely negates the fact that there is an '$a?$' not preceded one time unit earlier by an action. This implies that TPTL is undecidable, already in the pointwise semantics (when at least two clock variables are used).

From all these considerations, over finite runs, we get that in the pointwise semantics, MTL allows only to express behaviours of ICMETs, perfect channel machines require either the continuous semantics, or using MTL+Past or TPTL in the pointwise semantics. Applying the complexity results for channel machines recalled in proposition 4.9, we get most of the lower bounds that were announced (as said earlier, all these lower bounds results can be lifted to the infinite runs framework). The only case which is missing is the undecidability of MTL under the $\models^{\mathsf{point}}$-semantics.

***Remark* 4.10** We had mentioned that this presentation of the proof had been worked out while studying the control problem for MTL specifications. We have proven that, in that more general control (or two-player) framework, the backward matching property was expressible in MTL in the $\models_{\mathsf{f}}^{\mathsf{point}}$-semantics [BBC06a]. In that framework, we assume that the controller is playing against an adversary, and that adversary will be able to play a Check-action at most once in a play, exactly at the same time a read action '$a?$' occurs. Then, if globally the formula

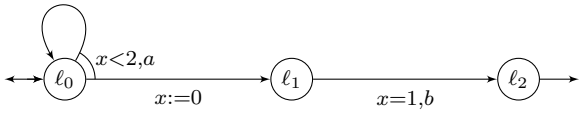
$$\mathbf{F}\,(a? \wedge \mathsf{Check}) \rightarrow \mathbf{F}\,(a! \wedge \mathbf{F}_{=1}\,\mathsf{Check})$$

is satisfied, it means that no matter when the adversary plays his Check-action, the above formula will hold. In other words, it means that there is no fake read actions along the computation, and thus that the backward matching property is satisfied. On the other hand, if the above formula is not satisfied, it means that there is at least one fake read action '$a?$', and that this was possible, only due to an insertion of an a on the channel. ⌟

4.4.2 ... but sometimes decidable, though

We now explain the decidability of MTL for the $\models_{\mathsf{f}}^{\mathsf{point}}$-semantics. It is well-known that LTL formulas can be transformed into alternating finite automata [MSS88, Var96]. In a similar way, we can transform any formula of MTL into an alternating timed automaton[4] with a single clock [OW05]. For instance, the formula $\mathbf{G}_{<2}\left(a \rightarrow \mathbf{F}_{=1}\,b\right)$ can be transformed into the following alternating timed automaton (where we label edges with actions):

[4]This model has been defined and studied independently in [LW05, LW08] and [OW05, OW07].

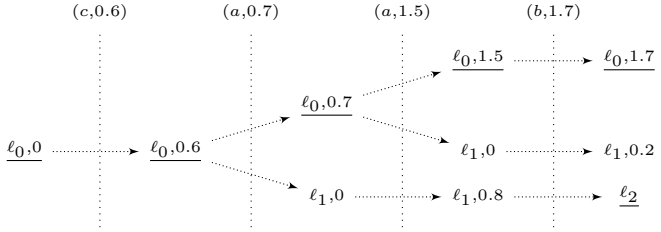


with the obvious interpretation that any time an a is done (within the two first time units), we fork a new thread which will check that a b appears one time unit later. Satisfiability of an MTL formula φ is then equivalent to the non-emptiness of the timed language accepted by the corresponding single-clock alternating timed automaton $\mathcal{B}_\varphi$. A timed word is accepted by an alternating timed automaton whenever there is an accepting behaviour (*i.e.* a tree) that reads that timed word.

Theorem 4.11 *Checking emptiness of alternating timed automata is decidable for one clock over finite timed words [LW05], any slight extension (infinite timed words, two clocks, ε-transitions) leads to undecidability [LW08].*

***Remark* 4.12** For a model-checking purpose, the above result is not sufficient. However only a slight extension of the proof of theorem 4.11 is needed to extend the result from satisfiability of MTL to the model-checking of MTL: to check that a timed automaton $\mathcal{A}$ satisfies the MTL property φ (under the $\models_{\mathsf{f}}^{\mathsf{point}}$-semantics), it is sufficient to check that the set of joint accepting behaviours of $\mathcal{A}$ and $\mathcal{B}_{\neg\varphi}$ is empty; And techniques used to prove the above theorem easily extend to that more general problem [OW05, OW07]. ⌟

We now briefly explain the decidability result in theorem 4.11. Consider the timed word $(c, 0.6)(a, 0.7)(a, 1.5)(b, 1.7)$. An execution of the previous alternating timed automaton on that timed word can be depicted as the following tree, which is not accepting as one of the branches (the second one on the picture) is not accepting (accepting states are underlined).



A configuration of the alternating timed automaton is a slice of the tree, for instance, $\{(\ell_0, 1.5), (\ell_1, 0), (\ell_1, 0.8)\}$ is a configuration. Because we consider finite words, there is no need to consider the tree structure of the execution, but we can reason globally on configurations of the automaton. There are

infinitely many such configurations, but as for the region automaton construction for timed automata [AD94], the precise values of the clocks is not really relevant, and the things which are important in a configuration are the integral parts of the clocks and the relative order of the fractional parts. For instance, for the above-mentioned configuration, we only need to know that there is a state $(\ell_0, 0)$ with fractional part 0, and two other states $(\ell_1, 0)$ and $(\ell_0, 1)$ such that the fractional part for $(\ell_1, 0)$ is larger than the fractional part for $(\ell_0, 1)$; This can be represented by a word that orders states with respect to their fractional part: $0 =_{\mathsf{frac}} (\ell_1, 0) \prec_{\mathsf{frac}} (\ell_0, 1.5) \prec_{\mathsf{frac}} (\ell_1, 0.8)$ is abstracted into the word $(\ell_1, 0) \cdot (\ell_0, 1) \cdot (\ell_1, 0)$, with the extra information that the first letter has fractional part 0.[5] For all configurations with the same abstraction, the possible future behaviours are the same, in a time-abstract bisimulation sense [OW07, LW08]. We can thus safely define an abstract transition system that will 'correspond' to the transition system of the original alternating timed automaton. Unfortunately, the set of abstract configurations of a single-clock alternating timed automaton is still infinite (because we cannot not bound *a priori* the size of a configuration, hence the length of the word that abstracts the configuration). However, it is not difficult to get convinced that there is a *well-quasi-order* on the set of abstract configurations (which is the sub-word preorder)! We can use this fundamental property to design an algorithm to decide the emptiness problem of single-clock alternating timed automata [FS01]. From that, we can derive an algorithm to decide the model-checking problem of MTL in the $\models_{\mathsf{f}}^{\mathsf{point}}$-semantics.

We will finally say few words on a possible encoding of the above abstract transition system into a channel machine (more precisely a slight extension of ICMETs with no more expressive power) [BMOW07]. From an abstract configuration, the immediate time successor can be obtained by turning around letters: roughly, the time successor of $(\ell_1, 0) \cdot (\ell_0, 1) \cdot (\ell_1, 0)$ (to distinguish how states move around, we use colors) is $(\ell_1, 1) \cdot (\ell_1, 0) \cdot (\ell_0, 1)$ (by delaying, the state concretizing the right-most $(\ell_1, 0)$ reaches the integer 1, its fractional part becomes the smallest one, hence it changes into $(\ell_1, 1)$ and moves to the left of the word). The next time successor is $(\ell_0, 2) \cdot (\ell_1, 1) \cdot (\ell_1, 0)$ (for the same reason, the state concretizing $(\ell_0, 1)$ reaches integer 2, hence we increase the integral part by 1 and move the letter to the left of the word). We can view this evolution as the behaviour of a channel which contains the encoding of the current configuration. It is worth noting that in this encod-

[5] Note that this is a simplified version of the real abstraction defined in [OW05].

ing, a complete *cycle* of the channel[6] corresponds to the elapsing of one time unit. Note that discrete transitions in the abstract transition system can also be implemented using rules of a channel machine, provided we allow some extra features like global renaming and occurrence testing.[7] The emptiness test is used to check that a final configuration is accepting.

Note that we can prove the decidability of TPTL (under the same semantics restrictions) with a single formula variable applying the same method.

Why does it not extend to infinite runs? The algorithmic idea that we have just presented to solve the model-checking problem for MTL in the $\models_{\mathsf{f}}^{\mathsf{point}}$-semantics cannot be lifted to the framework of infinite runs, because checking emptiness of single-clock alternating timed automata over infinite timed words is undecidable [LW08]. Indeed, in that case, we can no more reason on slices of a tree execution, as we need to check the (say Büchi) acceptance condition on every branch of the tree. Moreover a construction *à la* Miyano-Hayashi [MH84] cannot be used either, because the number of elements at a given level of an execution tree is potentially unbounded (because of the value of the clock), and it is not possible to use some well-quasi-order on slices as in the case of finite words to get a termination argument. Finally there is no way to circumvent the difficulty, because it has been proven in [OW06a] that the model-checking problem for MTL over infinite runs in the pointwise semantics is undecidable. We will not describe the undecidability proof, which relies on a reduction to and from the recurrence problem for ICMETs, and that problem is undecidable.

4.5 Some interesting fragments of MTL

To circumvent the untractability of MTL, several fragments thereof have been considered. We briefly describe some of those fragments, and give hints to explain why they are interesting and why they can be used for model-checking purposes. All the works I have made on that topic are joint with Nicolas Markey, Joël Ouaknine and James Worrell. In this section, we focus on the interpretation of the logics over infinite runs, all upper bounds can of course be lifted to the case of finite runs.

[6] A cycle of the channel corresponds to a portion of an execution along which everything that is written on the channel is read. Along an execution, the number of cycles made by the channel can be measured using an extra symbol that is put at the tail of the channel as soon as it is read from the head of the channel.

[7] We omit the definition of those notions here, because they will not be used, and better refer to [BMOW07].

4.5.1 The logic MITL

In 1991, the fragment MITL [8] of MTL has been proposed [AFH91, AFH96], which basically disallows punctual constraints on **U**-modalities. For instance, the formula '$\mathbf{G}\ (a \rightarrow \mathbf{F}_{=1}\, b)$' is not in MITL (because of the equality constraint '$=1$' on the $\mathbf{F}$-modality), whereas '$\mathbf{G}\ \big(a \rightarrow \mathbf{F}_{[1,2]}\, b\big)$' is in MITL. One reason why a positive result for MITL would be very satisfactory is that, in real timed systems, it is impossible to check a punctual constraint (because of the inherent imprecision of real systems). And disallowing punctual constraints leads indeed to an incredible improvement in the complexity of the model-checking problem!

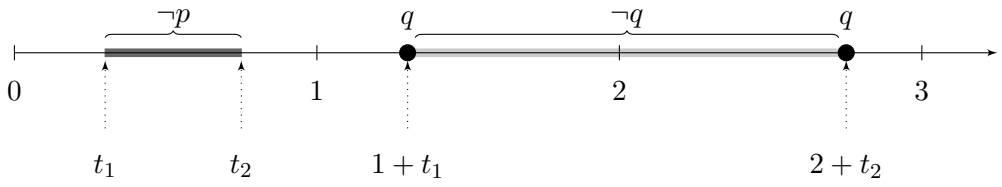
Theorem 4.13 ([AFH96]) *The model-checking and satisfiability problems*[9] *for* MITL *under the* $\models^{\mathsf{cont}}$*-semantics are* EXPSPACE*-complete.*

The reason for this fairly low complexity (compared to the NPR or even undecidable lower bound for MTL) is that the variability of models of MITL formulas can be controlled, and any property expressible in MITL can be recognized by a timed automaton (hence is somehow regular).

***Example* 4.14** We give an example (taken out of [AFH96]) to explain this regularity. Consider the MITL formula

$$\varphi \ = \ \mathbf{G}_{(0,1)}\, (p \rightarrow \mathbf{F}_{[1,2]}\, q)$$

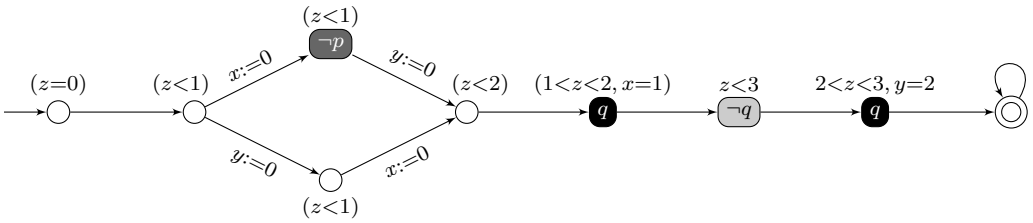
This formula says that within the first time unit, whenever p holds, then q must hold at some point between 1 and 2 time units later. To check the truth of this formula, a solution would be, each time p holds within the first time unit, to start a clock and check that within 1 to 2 time units from that point, q becomes true. However, there may be an unbounded number of segments in which p holds within the first time unit. Hence, applying this method, an unbounded number of clocks would *a priori* be required. A more clever method needs to be used, which is illustrated below.



[8] Standing for 'Metric Interval Temporal Logic'.

[9] Also in that case, the two problems are equivalent, because MITL is closed under negation, and as already written in footnote 2 page 54, the behaviours of a timed automaton can be captured by a polynomial size MITL formula.

We focus on models where q does not hold at time 2, and where there is a last date within the interval $(1, 2)$ at which q holds, and a first date within the interval $(2, 3)$ at which q holds. Other cases can be handled separately in a very similar manner. The idea is to point to the last time q holds in the interval $(1, 2)$ and the first time q holds in the interval $(2, 3)$. Then, the only points within $(0, 1)$ that will not satisfy $\mathbf{F}_{[1,2]}\, b$ are in the interval (t_1, t_2), where $t_1 + 1$ is the point distinguished in $(1, 2)$ and $t_2 + 2$ is the point distinguished in $(2, 3)$. Hence, the only chance for the global formula to hold from the beginning is that there is no p in the interval (t_1, t_2). We can then build a timed automaton which guesses time points t_1 and t_2, resets clocks at those time points, and checks that those time points really satisfy the expected properties (for completeness, the constructed automaton is given below).



⌟

The algorithm developed in [AFH96] relies on ideas illustrated in the previous example, and from a formula $\varphi \in \mathsf{MITL}$, builds a timed automaton $\mathcal{B}_\varphi$ (with a Büchi acceptance condition) such that for every timed automaton $\mathcal{A}$, $\mathcal{A} \models \neg\varphi$ if and only if the classical product $\mathcal{A} \times \mathcal{B}_\varphi$ has no accepting run.

4.5.2 The logic Safety-MTL

The logic Safety-MTL has been proposed in [OW05] as a logic allowing to express safety properties like bounded response time properties of the form '$\mathbf{G}\,(a \to \mathbf{F}_{=1}\, b)$' or '$\mathbf{G}\,(a \to \mathbf{F}_{\leqslant 5}\,(b \wedge \mathbf{F}_{=1}\, c))$'. Unlike MITL, it partly allows punctual constraints on modalities, but cannot express general response time properties. Roughly, a formula of MTL is in Safety-MTL whenever every positive instance[10] of an $\mathbf{U}$-modality is constrained by a bounded interval.

Unlike all the logics we have mentioned so far, Safety-MTL is not closed under negation. Hence, the model-checking and the satisfiability problems need to be distinguished.

Theorem 4.15 ([OW05]) *The model-checking problem for* Safety-MTL *under the* $\models^{\mathsf{point}}$*-semantics is decidable.*

[10] *I.e.*, every instance under the scope of an even number of negations.

The reason for the decidability of the model-checking problem for this logic is that it can only express 'safety' properties, that is properties whose negations, if violated, are violated after a finite prefix. For those formulas, we can thus build a single-clock alternating timed automaton over finite words and with only accepting states, which recognizes all bad prefixes of the formula. Then the encoding for proving the decidability of alternating timed automata over finite timed words can be used in that case as well.

Unlike MITL, Safety-MTL is not closed under negation, we can thus not lift the above result to the satisfiability, and it needs to be proven separately.

☞ **Theorem 4.16** *The satisfiability problem for* Safety-MTL *under the* $\models^{\mathsf{point}}$*-semantics is decidable [OW06b] but non-elementary [BMO+08].*

The non-termination problem [11] in ICMETs can be reduced to the satisfiability of formulas in Safety-MTL, the first problem being proven non-elementary [BMO+08].

4.5.3 The logics coFlat-MTL

From the investigations so far, it turns out that punctuality is the core of the undecidability, and this was somehow admitted and not really problematic, because punctual constraints are not really realistic, because hardly satisfiable by a real timed systems, where there is some inherent imprecision. Thus, the first fragment we will present has maybe more a theoretical interest than a practical motivation, in that we will present a fragment of MTL, where punctuality is allowed, but of course with some other restrictions. The second fragment will unify the approaches of MITL and of the first fragment, providing an expressive logic that can be rather efficiently model-checked! These two fragments are called in the original papers coFlat-MTL [BMOW07, BMOW08]. In this subsection, to distinguish between these two logics, we will call the first fragment coFlat-$\mathsf{MTL}_{\mathsf{LTL}}$, and the second fragment coFlat-$\mathsf{MTL}_{\mathsf{MITL}}$, for reasons that will become obvious very soon.

The fragment coFlat-$\mathbf{MTL}_{\mathbf{LTL}}$. The syntax of the logic coFlat-$\mathsf{MTL}_{\mathsf{LTL}}$ restricts the $\varphi \mathbf{U}_I \psi$-subformulas in such a way that if I is not bounded, then ψ must be in LTL, whereas φ can be in coFlat-$\mathsf{MTL}_{\mathsf{LTL}}$ itself. It is worth noticing that coFlat-$\mathsf{MTL}_{\mathsf{LTL}}$ is rather powerful as it contains LTL, Bounded-MTL (the subset of MTL where all modalities are bounded), and is closed under invariance, which is rather useful for specifying correctness

[11] The termination problem asks whether all executions are finite, the non-termination problem thus asks whether there is an infinite execution.

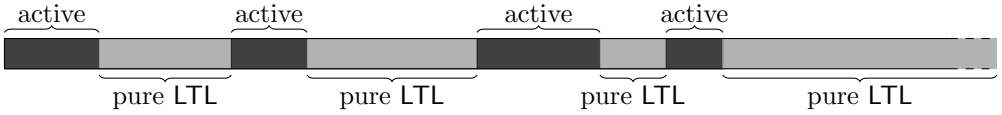
properties in critical systems. In particular, the formula '$\mathbf{G}\ (a \rightarrow \mathbf{F}_{=1}\, b)$' is in coFlat-MTL$_{\mathsf{LTL}}$, but '$\mathbf{F}\, \mathbf{G}_{\leqslant 1}\, a$' is not in coFlat-MTL$_{\mathsf{LTL}}$. Using techniques completely different from those developed for MITL, we have proven the following result:

☞ **Theorem 4.17 ([BMOW07])** *The model-checking problem for* coFlat-MTL$_{\mathsf{LTL}}$ *under the* $\models^{\mathsf{point}}$*-semantics is* EXPSPACE*-complete.*

The decidability and upper bound are rather involved, and rely on an encoding into a slight extension of channel machines. We have already mentioned that the halting problem for channel machines is undecidable (Proposition 4.9). However, we will not reduce to the general halting problem, but to the halting problem when we bound the number of cycles of the whole channel (*cf.* footnote 6 page 62). This restricted problem is proven to be solvable in space polynomial in the size of the channel machine and in the value of the cycle bound in [BMOW07].

We have already mentioned that we could encode MTL formulas into single-clock alternating timed automata, and thus reduce the model-checking problem to checking that there is (or not) a joint behaviour in a product of a timed automaton and a single-clock alternating timed automaton corresponding to the negation of the formula (see subsection 4.4.2). We have also mentioned that this question could be reduced to the halting problem of some channel machine. In this encoding, one cycle of the channel corresponds to the elapsing of one time unit. If we consider a formula φ of Bounded-MTL (fragment of MTL where all modalities are labelled by a bounded interval), it is rather clear that its verification along a run ϱ only requires to look at a time-bounded prefix of ϱ (the time bound can be set as the sum of all the upper bounds of intervals labelling the modalities in the formula). Hence, in the translation into channel machines, it means that we can restrict to executions with a bounded number of cycles, which suggests an algorithm for deciding the model-checking of Bounded-MTL.

We extend this idea to coFlat-MTL$_{\mathsf{LTL}}$, and because of the syntactic restriction made in coFlat-MTL$_{\mathsf{LTL}}$, if φ is a coFlat-MTL$_{\mathsf{LTL}}$ formula, the alternating automaton $\mathcal{B}_{\neg\varphi}$ has a special structure (that is somehow 'flat' — hence the name for coFlat-MTL$_{\mathsf{LTL}}$, because the negation of a formula in that logic is flat). An execution not satisfying a formula φ in coFlat-MTL$_{\mathsf{LTL}}$ can be decomposed as follows:



where the number of active fragments is at most exponential and the total duration of active fragments is also at most exponential. An active fragment corresponds to a cycle-bounded computation in a channel machine, whereas the pure LTL parts are very simple computations corresponding to those of a finite automaton (only simple LTL formulas are checked).

***Remark* 4.18** Unlike MITL, coFlat-MTL$_{\mathsf{LTL}}$ does not only express 'regular' properties. For instance, the property '$\mathbf{G}\,(a \to \mathbf{F}_{=1}\, b)$', which belongs to coFlat-MTL$_{\mathsf{LTL}}$, can be used to generate the context-free language $\{a^n b^m \mid n \leqslant m\}$. ⌟

***Remark* 4.19** Also, note that coFlat-MTL$_{\mathsf{LTL}}$ might enforce a rather high *variability*.[12] For instance, the formula

$$\varphi_n = a \wedge \mathsf{Double} \wedge \mathbf{G}_{[0,2^n)}\, \mathsf{Double}$$

where

$$\mathsf{Double} = \Big(b \to \mathbf{F}_{=1}\,(b \wedge \mathbf{X}_{<1}\, a)\Big) \wedge \Big(a \to \mathbf{F}_{=1}\,(b \wedge \mathbf{X}_{<1}\, a)\Big)$$

enforces a model to have at least a doubly-exponential variability (in n). ⌟

Finally, we conclude by noticing that coFlat-MTL$_{\mathsf{LTL}}$ is not closed under negation. Though this is not formally stated in the mentioned paper, the following undecidability result is a straightforward consequence of the undecidability proof for MTL in the $\models^{\mathsf{point}}$-semantics.

Theorem 4.20 ([OW06a]) *The satisfiability problem for* coFlat-MTL$_{\mathsf{LTL}}$ *under the* $\models^{\mathsf{point}}$*-semantics is undecidable.*

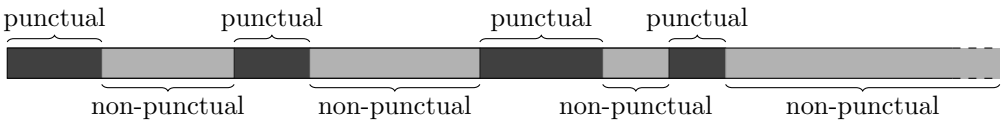
The fragment coFlat-MTL$_{\mathsf{MITL}}$. Recently we have extended the fragment coFlat-MTL$_{\mathsf{LTL}}$ by relaxing the restriction that a sub-formula on the right of an unbounded 'Until' modality be in LTL in the fragment: the fragment coFlat-MTL$_{\mathsf{MITL}}$ now allows such formulas to be in MITL instead. This new fragment is a superset of both MITL and coFlat-MTL$_{\mathsf{LTL}}$, but it can be model-checked rather efficiently!

☞ **Theorem 4.21 ([BMOW08])** *The model-checking problem for* coFlat-MTL$_{\mathsf{MITL}}$ *under the* $\models^{\mathsf{cont}}$*-semantics is* EXPSPACE*-complete.*

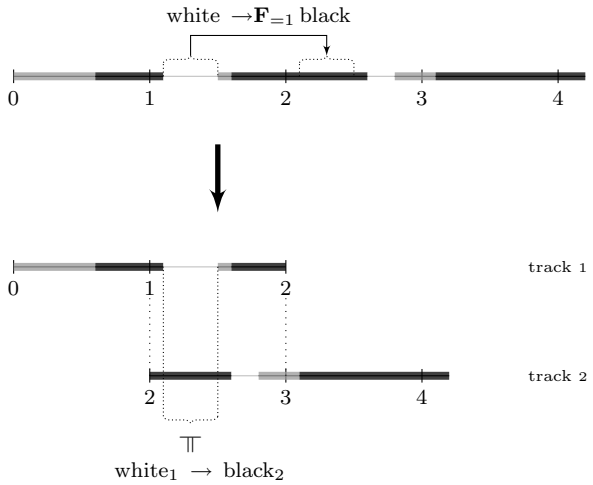
The idea of the proof mixes the approaches for MITL and for coFlat-MTL$_{\mathsf{LTL}}$. Alternating timed automata corresponding to negations of coFlat-MTL$_{\mathsf{MITL}}$ formulas do not yield anymore interesting properties for the models thereof.

[12] The variability of a run is the maximal number of discrete transitions made within one time unit along that run.

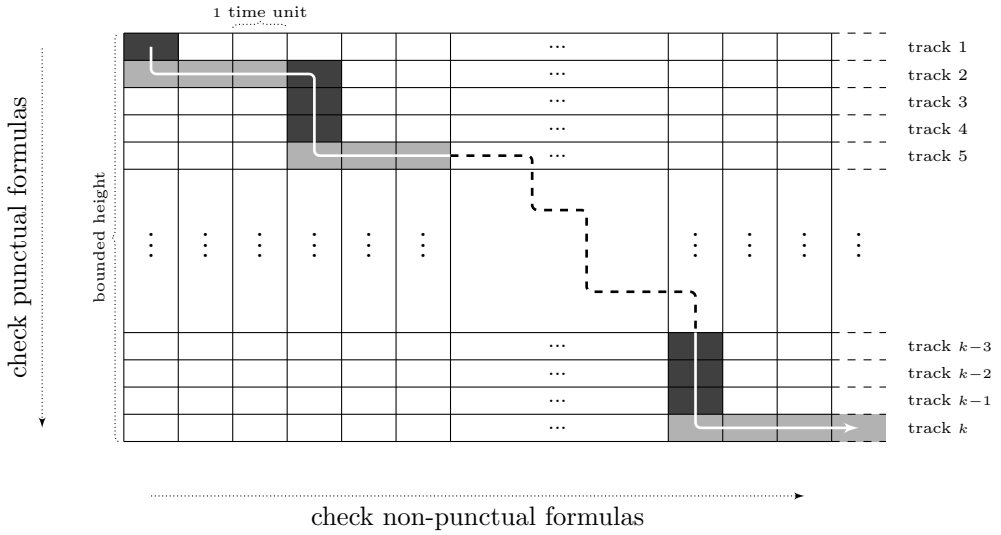
Hence to prove the above theorem, we do not use anymore an automata-theoretic but a purely-logical approach. Because of the syntactical restrictions made in coFlat-MTL$_{\mathsf{MITL}}$, we can still decompose a signal not satisfying an coFlat-MTL$_{\mathsf{MITL}}$-formula φ as follows:



where the number of 'punctual' fragments can be bounded by an exponential, and the global duration of 'punctual' fragments can also be bounded by an exponential. Roughly, a non-punctual fragment does not check any punctual formulas, hence we will be able (i) to bound the variability of the model (following MITL ideas), and (ii) to stretch (in a reasonable way) those parts of the model, and (iii) forget the timing constraints (by adding 'tick' atomic propositions, that will be sufficient to recover the timing constraints). A punctual fragment cannot be treated in the same way, as it is not possible to stretch the model without changing the truth of the formula, but fortunately, the global duration of the punctual fragments is small, and we will use another trick. Assume that we have to check that the formula 'white $\rightarrow \mathbf{F}_{=1}$ black' holds within the interval $[1, 2]$ of the signal below (where colors represent atomic propositions). Then we will stack the two time units $[1, 2]$ and $[2, 3]$ on two different tracks, and check that color white on track 1 implies color black on track 2. That way we have removed the punctual modality $\mathbf{F}_{=1}$.



We do the same for all the fragments with punctual modalities, and transform our problem into a 'tableau' satisfiability problem for LTL+Past (the extension of LTL with past-time modalities), as illustrated below.

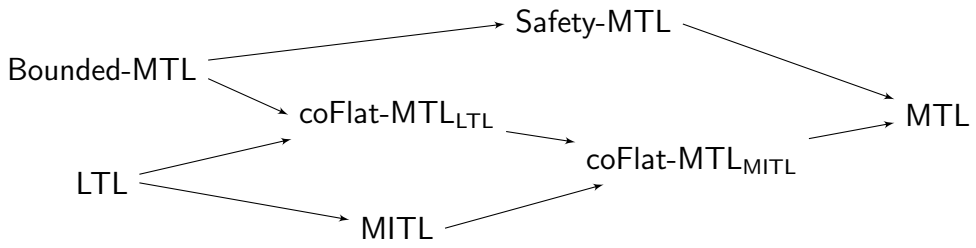


On a track of the tableau, only non-punctual constraints are checked (thanks to stretching, timing constraints can be abstracted and expressed in LTL). Columns of the tableau can be used to check punctuality constraints: extending the ideas illustrated before, when punctuality constraints need to be checked, we stack two consecutive time units, so that two events separated by exactly 1 time unit will be on the same vertical line, but the second event will be one track below; Hence, a constraint '$\mathbf{F}_{=2}\,\varphi$' on track i at time t will be propagated as a constraint '$\mathbf{F}_{=1}\,\varphi$' on track $i+1$ at time t, and finally as a constraint 'φ' on track $i+2$ at time t; We can then forget about timing constraints and abstract them using propagating rules; Globally, we need past-time modalities (think for instance of a formula $\varphi\,\mathbf{U}_{=1}\,\psi$ that needs to be checked). Thus we construct an LTL+Past formula that has a 'tableau' model (*i.e.*, a model with several tracks, as we have depicted above) if and only if the original coFlat-$\mathsf{MTL}_{\mathsf{MITL}}$-formula has a model. Then, a model for the latter can be extracted from the tableau following the snake-line (when there are punctual constraints (black parts), we continue to the bottom, and otherwise (grey parts) we continue to the right). The number of tracks that are necessary can be bounded, thanks to the decomposition of a model into punctual and non-punctual fragments.

We have thus reduced (using an exponential construction) our problem to the satisfiability problem for LTL+Past, which is known to be solvable in PSPACE [Rey04].

4.5.4 Summary of the expressiveness and complexity results

In this section, we have presented several fragments of MTL that have been considered in the literature.[13] We summarize the expressiveness of the different fragments on the following picture, where an arrow $\mathcal{L} \to \mathcal{L}'$ means that $\mathcal{L}'$ is more expressive than $\mathcal{L}$. Note that all inclusions are syntactical inclusions, and though we did not prove it formally, we claim that all these expressiveness results are strict (probably techniques similar to those developed in [BCM05] could be used, but this is always tedious to do, and we did not take the time to do so).



For the model-checking and satisfiability problems, we have focused on infinite runs, but any upper bound can be lifted to the framework of finite runs. In the table that summarizes the different complexity results, we are a bit more precise than we have been so far in this section, and we distinguish the cases when constants used in formulas are encoded in unary or in binary (because that actually makes a difference). In grey, we give complexity results that can be deduced from the mentioned references, but that are not formally stated. The model-checking problem for LTL is PSPACE-complete (see theorem 2.12). The logic MITL is closed under negation and can express the behaviours of timed automata, hence the complexities that are given are those for the two problems. The logic Bounded-MTL is closed under negation but cannot express behaviours of timed automata, hence we have several gaps in the table. For the three last logics, we also need to distinguish between the two problems. This distinction is made in the right-most column.

[13] We do not include in this summary the different fragments that have been studied by Hirshfeld and Rabinovich, for instance in [HR05], as they do not compare easily to the fragments we have studied.

	$\models^{\mathsf{point}}$/binary	$\models^{\mathsf{point}}$/unary	$\models^{\mathsf{cont}}$/binary	$\models^{\mathsf{cont}}$/unary	
LTL	PSPACE-c. [SC85]		PSPACE-c. [Rey04]		
MITL	EXPSPACE-c. [AFH96]	PSPACE-c. [HR05]	EXPSPACE-c. [AFH96]	PSPACE-c. [HR05]	
Bounded-MTL	EXPSPACE-c. [BMOW07]	EXPSPACE [BMOW07]	?	?	m.-c.
	EXPSPACE-c. [BMOW07]	PSPACE-c. [BMOW08]	EXPSPACE-c. [BMOW08]	PSPACE-c. [BMOW08]	sat.
coFlat-MTL$_{\mathsf{LTL}}$	EXPSPACE-c. [BMOW07]	EXPSPACE-c. [BMOW08]	EXPSPACE-c. [BMOW08]	EXPSPACE-c. [BMOW08]	m.-c.
	undecidable [OW06a]				sat.
coFlat-MTL$_{\mathsf{MITL}}$	EXPSPACE-c. [BMOW08]		EXPSPACE-c. [BMOW08]		m.-c.
	undecidable [OW06a]				sat.
Safety-MTL	decidable [OW05]		?		m.-c.
	decidable [OW06b] non-elementary [BMO$^+$08]		? non-elementary [BMO$^+$08]		sat.

It is worth noting that several fragments have rather appealing complexities (indeed, in the framework of timed systems, we seldom find better complexities than PSPACE).

4.6 Conclusion and further work

In this section, we have studied popular linear-time timed temporal logics, and in particular the expressiveness of various such logics. We have also considered decidability and complexity aspects for the model-checking and the satisfiability of those logics. The investigations for the full natural extensions were really pessimistic, with a single decidability result (and very high complexity), furthermore under a rather restrictive semantics. Hence, the quest for restrictions yielding better complexity results was a natural line of research, and several fragments have been defined with, in most cases, a really harsh improvement in the complexities. In those fragments, one can express many interesting properties, like invariance properties, bounded response time, *etc.*

Following the development of chapter 3, I would be interested in being involved in the development of a tool for fragments of linear-time timed temporal logics. In particular, it seems that developing good data structures for model-checking those logics is really challenging, as basic data structures like DBMs will *a priori* not be sufficient. However, there is one point that should be mentioned: even though the model-checking of MTL is hard, truth of MTL formulas is invariant within a region (that can be easily seen for the pointwise semantics using the transformation into alternating timed automata), hence we do not need general data structures like polyhedra. The difficult point is then that a large number of timing constraints need to be stored during the model-checking process, but I think that DBMs with a number of variables that may change dynamically could be used. And as far as I know, the current version of Uppaal already implements DBMs with a number of variables which evolves dynamically. However, there might be a more clever data structures for those fragments that have a rather low complexity. Maybe we could use channel machines as a data structure in that case? Indeed the channel of a channel machine can store an unbounded amount of timing information.

Together with Mark Jenkins (Oxford University, England), Joël Ouaknine and James Worrell, we are currently investigating whether the flatness ideas we have defined here could be used in the framework of branching-time logics. Indeed, the satisfiability problem for TCTL is known to be undecidable in general [ACD93]. We think that flatness assumptions could lead to a rather impressive improvement of the satisfiability problem (like going down to 2EXPTIME). Following the techniques we have used for coFlat-$\mathsf{MTL}_{\mathsf{LTL}}$, this should require studying alternating channel machines with a bounded number of cycles.

Chapter 5

Weighted timed automata, a model for embedded systems

5.1 Introduction

New challenges in embedded systems verification. Since a couple of years, the verification technology for timed automata has evolved in several interesting directions, to answer new challenges posed by modern real-life systems, like the control of resources (*eg.* energy consumption, bandwidth or memory limitations, *etc.*). Hybrid systems [Hen96, HKPV98, Ras05] can be seen as a first answer to these new challenges, but they do not enjoy very nice decidability properties, and even though many works are devoted to that model (with the developments of heuristics, approximation algorithms, extraction of decidable subclasses, *etc.*), there might be other alternative and more accurate models. In that direction, weighted (or priced) timed automata [ALP01, BFH$^+$01] have been designed as an extension of the timed automaton formalism, which uses observer variables to measure the performance or the cost of executions of the system. Several systems can be modelled using weighted timed automata — we can for instance mention scheduling problems [RLS06, BLR05a], or production systems, see [BLR05a] for examples of systems that can be modelled using weighted timed automata — and natural interesting optimization questions can be asked.

Weighted/priced timed automata. Weighted timed automata[1] extend classical timed automata with cost information both on locations and transitions of the automaton. The cost labelling a location represents the cost

[1] In this chapter, we follow the terminology of [ALP01], but this model is called priced timed automaton in [BFH$^+$01].

per time unit that has to be paid for staying in that location, and the cost labelling a transition represents the cost to be paid for taking the transition. That way, every run in the automaton has a global cost, which is the accumulated cost of every delay or discrete move along the run. Note that the value of the cost variable does not restrict the possible executions of the system, but gives a quantitative information on the quality of an execution.

Model-checking and games with an optimization criterion. Given that the cost variable gives a measure of the quality of an execution, we can use that model to optimize the performance of a system. For instance, one may want to optimize the cost for reaching some distinguished set of locations, and to synthesize schedules to achieve that optimal cost (this is for instance the case for scheduling problems, where one wants to schedule tasks on machines, in such a way that the energy consumption be minimal), or one may want to compute the optimal mean-cost and schedules achieving that optimal mean-cost.

More generally we can consider various model-checking problems where the cost, viewed as an observer variable, can be used to specify properties of the system. In that context, we will consider natural extensions of temporal logics, where modalities are decorated with constraints on the cost. For instance, in WCTL, an extension of CTL with cost constraints, one can write properties like '$\mathbf{A}\,\mathbf{G}\ (\mathsf{request} \rightarrow \mathbf{A}\,\mathbf{F}_{\leqslant 5}\,\mathsf{ack})$', which says that a request is always acknowledged, and it does not cost more than 5.

Everything we have presented so far concerns closed systems, where we can control everything in the system. In general, we will be interested in embedded systems, which execute in an environment: in that case we say that the systems are *open*. Classically, those systems are modelled as two-player games [Thom02], and the optimization question(s) we asked then becomes 'can we synthesize a strategy for the system (or controller) so that the cost is the best we can expect, whatever is the behaviour of the environment?'.

Content of this chapter. These last years, there has been a prolific literature on that model. In section 5.2, we will present the model of weighted timed automata. Then in section 5.3, we will present the first decidability results that have been obtained for basic optimization problems. In section 5.4, we investigate the optimization problems in the context of open systems. In section 5.5, we focus on more general model-checking problems, where properties are expressed in an extension of temporal logics with cost constraints. Finally, in section 5.6, we give some recent investigations and perspectives.

5.2 The weighted timed automaton model

In this section we introduce the weighted (or priced) timed automaton model, that has been proposed in 2001 for representing resource consumption in real-time systems [ALP01, BFH+01]

Definition 5.1 *A* weighted (or priced) timed automaton *is a tuple*

$$\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L}, \mathsf{cost})$$

where $(\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L})$ *is a (diagonal-free) timed automaton, and* $\mathsf{cost} : L \cup E \to \mathbb{N}$ *is a* cost function *which assigns a value to each location and to each transition. The cost (function)* cost *is said* stopwatch *whenever* $\mathsf{cost}(L) \subseteq \{0, 1\}$ *and* $\mathsf{cost}(E) = \{0\}$.

The semantics of a weighted timed automaton is that of the underlying timed automaton. The role of the cost functions is to give a quantitative information on the moves in the system. The value given to a location represents a cost rate, and delaying t time units in a location ℓ will cost '$t \cdot \mathsf{cost}(\ell)$'. The value given to a transition represents the cost of taking that transition. Formally, the cost of the two types of moves in a weighted timed automaton is defined as follows:[2]

$$\begin{cases} \mathsf{cost}\left((\ell, v) \xrightarrow{t} (\ell, v+t)\right) = t \cdot \mathsf{cost}(\ell) \\ \mathsf{cost}\left((\ell, v) \xrightarrow{e} (\ell', v')\right) = \mathsf{cost}(e) \end{cases}$$

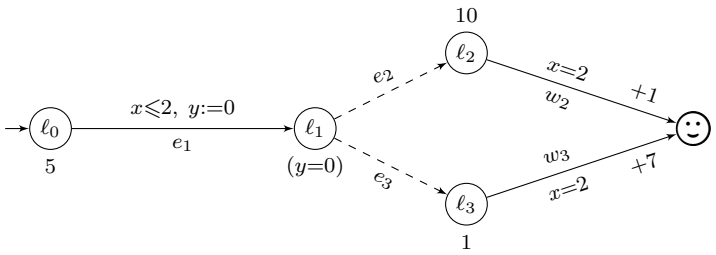
A *run* ϱ of a weighted timed automaton is a run of the underlying timed automaton, *i.e.*, a finite or infinite sequence of moves in the transition system (with a strict alternation of delay and discrete moves). The cost of ϱ, denoted $\mathsf{cost}(\varrho)$, is the sum of the costs of all the simple moves along ϱ.

***Example* 5.2** We consider the weighted timed automaton $\mathcal{A}$ depicted on the next figure, where we forget about dashed and plain transitions for now. When relevant (*i.e.*, when the cost is non-null), we decorate each location with a value (like 5 for location ℓ_0), that represents the cost rate in that location, and we decorate each transition with a value (like +7 for transition w_3), that represents the discrete cost of firing that transition. A possible run in $\mathcal{A}$ is:

$$\varrho \;=\; (\ell_0, 0) \xrightarrow{0.1} (\ell_0, 0.1) \xrightarrow{e_1} (\ell_1, 0.1) \xrightarrow{e_3} (\ell_3, 0.1) \xrightarrow{1.9} (\ell_3, 2) \xrightarrow{w_3} (\text{☺}, 2)$$

The cost of ϱ is $\mathsf{cost}(\varrho) = 5 \cdot 0.1 + 1 \cdot 1.9 + 7 = 9.4$ (the cost per time unit is 5 in ℓ_0, 1 in ℓ_3, and the cost of transition w_3 is 7).

[2]Note that we overload the notation cost, which designs both the cost assigned to a transition or a location in a weighted timed automaton, and the cost assigned to a move in the transition system. Later it will also represent the cost of an execution.



⌟

***Remark* 5.3** In our model, we have only defined one cost function. In general, there can be several cost functions that measure the quality of runs. When this is useful, we will say that several cost functions can (or need to) be used. ⌟

5.3 Optimization problems

Unlike hybrid systems, cost variables do not constrain the behaviours of the system, but are *'observer variables'*: they give a quantitative information on the quality of a run, and can be used to measure the performance of a system. Several optimization criteria can then be thought of, like the optimal cost for reaching some goal in the system, or the optimal mean-cost that can be achieved along infinite executions of the system. These optimization problems are relevant for instance in scheduling problems, where the cost evolution can be viewed as resource consumption.

5.3.1 Decidability results

In this subsection we give an overview of the decidability and complexity results for the optimization problems that we have briefly mentioned before. In the next subsection we will give a rough idea why these results hold.

The optimal cost problem. Intuitively, the *optimal cost problem* asks what is the optimal cost for reaching the goal locations in a weighted timed automaton. We assume $\mathcal{A} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathsf{cost})$ is a weighted timed automaton. The optimal cost for reaching goal locations in $\mathcal{A}$ is defined as:

$$opt\_cost_{\mathcal{A}} = \inf\{\mathsf{cost}(\varrho) \mid \varrho \in \mathsf{Runs}_{\mathsf{f}}^{\mathsf{acc}}(\mathcal{A}, s_0)\}$$

By extension when we will speak of the complexity, we will mean the complexity of the corresponding decision problem, which asks, given a threshold $c \in \mathbb{Q}_{\geqslant 0}$, whether $opt\_cost_{\mathcal{A}} \leqslant c$. If $\varepsilon > 0$, a run $\varrho \in \mathsf{Runs}_{\mathsf{f}}(\mathcal{A}, s_0)$ is an *ε-optimal schedule* in $\mathcal{A}$ if $opt\_cost_{\mathcal{A}} \leqslant \mathsf{cost}(\varrho) \leqslant opt\_cost_{\mathcal{A}} + \varepsilon$.

In this context, the first problem which has been solved already in the early 90's by Courcoubetis and Yannakakis is the *optimal time problem*, where the cost represents the time that has elapsed (the cost rates in locations are equal to 1 — they increase at the same speed as the time — and discrete costs of transitions are set to 0): the problem then consists in computing the optimal time for reaching one of the distinguished goal locations in a timed automaton.

Theorem 5.4 ([CY92]) *The optimal time in timed automata is computable in exponential time.*

Applying the further results (theorem 5.6) on weighted timed automata, we can refine this result, and computing the optimal time in timed automata can actually be solved in polynomial space. Moreover, we can prove that the problem is indeed PSPACE-complete (if there is an answer to the reachability problem, we can bound the duration of a witness run by an exponential, and then answering positively to the decision problem for that upper bound duration is equivalent to answering the reachability question, which is known to be PSPACE-hard).

Almost ten years after this first result, the general cost optimal reachability problem in weighted timed automata has been formulated and solved independently in [ALP01] and in [BFH+01].

Theorem 5.5 ([ALP01, BFH+01]) *The optimal cost in weighted timed automata is computable (in exponential time).*

The algorithm developed in [ALP01] is based on an extension of the classical region automaton, and yields an EXPTIME upper bound for solving the problem, whereas the algorithm developed in [BFH+01] is based on well-quasi-orders, which gives no good information on the complexity of the problem.

With Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin, we have computed the precise complexity of the problem.

☞ **Theorem 5.6 ([BBBR07])** *The optimal cost problem in weighted timed automata is* PSPACE*-complete. Furthermore, for every $\varepsilon > 0$, ε-optimal schedules can be computed.*

***Remark* 5.7** Note that the above result also holds when the costs of locations on transitions are taken in $\mathbb{Z} = \mathbb{N} \cup -\mathbb{N}$, the set of integers. ⌟

The optimal mean-cost problem. Intuitively, the *optimal mean-cost problem* asks what is the optimal mean-cost (*eg.* cost per time unit) that can be achieved (or approximated) in a weighted timed automaton. To define the most general mean-cost problem, we assume that $\mathcal{A}$ is a weighted timed automaton with **two** cost functions, say cost and reward. Then, the optimal mean-cost of $\mathcal{A}$ with respect to cost and reward is formally defined as:

$$opt\_cost^{\omega}_{\mathcal{A}} = \inf\{\mathsf{mean\_cost}(\varrho) \mid \varrho \in \mathsf{Runs}(\mathcal{A}, s_0)\}$$

where $\mathsf{mean\_cost}(\varrho)$ is defined as $\liminf_{n\to+\infty} \frac{\mathsf{cost}(\varrho_n)}{\mathsf{reward}(\varrho_n)}$ (ϱ_n is the prefix of length n of ϱ). We use the 'lim inf' operator because the limit might not be properly defined. A particular case is when the reward corresponds to the time elapsed, in which case the value $\mathsf{mean\_cost}(\varrho)$ is the mean cost per time unit along run ϱ. If $\varepsilon > 0$, a run $\varrho \in \mathsf{Runs}(\mathcal{A}, s_0)$ is an *ε-optimal schedule* in $\mathcal{A}$ if $opt\_cost^{\omega}_{\mathcal{A}} \leqslant \mathsf{mean\_cost}(\varrho) \leqslant opt\_cost^{\omega}_{\mathcal{A}} + \varepsilon$.

With Ed Brinksma and Kim G. Larsen, we have proven the following result.

☞ **Theorem 5.8 ([BBL04, BBL08])** *Under some restrictions for the reward function, the optimal mean-cost problem is* PSPACE*-complete in weighted timed automata. Furthermore, for every $\varepsilon > 0$, ε-optimal schedules can be computed.*

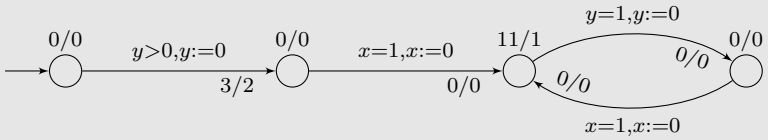
***Remark* 5.9** The restrictions mentioned in the above theorem assume the function reward be strictly non-Zeno, *i.e.*, along any cycle of the region automaton, the reward increases by some positive lower-bounded amount. If we consider the time elapsed instead of a general reward-function, this amounts to the classical strongly non-Zeno hypothesis, that is for instance made in [AMPS98].

Note that this hypothesis is required to get the above result, as we have exhibited a counter-example to our algorithm in which this hypothesis is not satisfied, see table 5.1. ⌟

5.3.2 The corner-point abstraction

The two decidability results mentioned in the previous subsection can be proven using a refinement of the region construction. Indeed, regions are not suitable for computing optimal (mean-)costs because costs of region-equivalent trajectories may have pretty different costs. For example, the cost of run ϱ given in example 5.2 is 9.4 whereas the cost of the (region-equivalent) run delaying 0.9 time units in ℓ_0 and then 2.1 time units in ℓ_3 is 13.6. However we are not interested in computing the costs of all possible runs, but rather

We consider the following weighted timed automaton. We write α/β to indicate the cost and reward of the locations and the edges. We can notice that the reward is not strongly non-Zeno, because of the right-most cycle.



In this automaton, for every (infinite) run ϱ, $\mathsf{mean\_cost}(\varrho) = +\infty$, whereas the algorithm (based on the corner-point abstraction, see next subsection) computes 2.

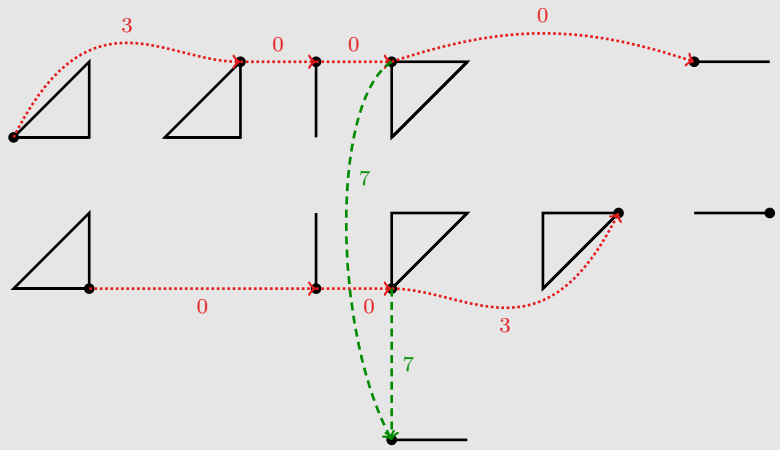
Table 5.1: The reward needs to be strongly non-Zeno

to compute extremal (*i.e.*, minimal and/or maximal) cost values. The idea is then to record the cost of moving through extremal points of the regions (which have integral coordinates). These points are called *corner-points*, and will 'decorate' regions. We build a graph, called the *corner-point abstraction*, which refines the classical region automaton, and whose states are tuples (ℓ, R, α) where ℓ is a location of the original automaton, R is a region, and α is a corner-point of R. Intuitively, being in state (ℓ, R, α) of this graph means that we are in location ℓ, in region R, close to the extremal point α. We illustrate this notion of corner-points in example 5.10, and informally explain how we build the corner-point abstraction.

***Example* 5.10** We illustrate the notion of corner-points in a two-dimensional clock space. As recalled in table 2.2 (page 23), classical evolution of regions can be schematized as in table 5.2: when time elapses, regions are visited following time successors (the immediate successor of a triangular region is a flat region while the immediate successor of a flat region is a triangular region), and when firing transitions, clocks may be reset, and regions are then somehow projected into flatter regions.

The corner-point abstraction is depicted in table 5.2. Corners decorating regions are indicated with a black bold dot. We consider the top-left-most region of the figure decorated with the corner in the bottom. When time elapses, it is transformed into the top corner of the same region which is almost one time unit later: thus, as the cost rate in the current location is supposed to be 3 per time unit, the cost of this move is 3 (because almost one time unit has elapsed, the cost has thus increased by almost 3). The next move is to enter the next region (which is flat) but to stay close to the same corner. The cost is thus almost 0 (because almost no time has elapsed), that is why we label the move by 0. And

The region abstraction



The corner-point abstraction

Table 5.2: Region *vs* corner-point abstraction

so on. For discrete moves, regions are transformed as usual, and corners are also projected (the projection preserves the property of extremal points of polyhedra). Transitions are then labelled with the cost of the transition (7 in our example). ⌟

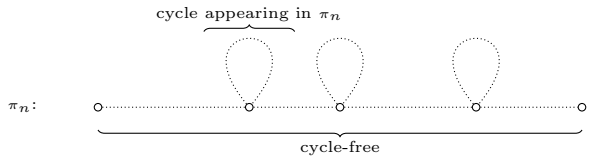
Given a weighted timed automaton $\mathcal{A}$, we build the so-called *corner-point abstraction of* $\mathcal{A}$, denoted $\mathsf{CP}(\mathcal{A})$, which refines the classical region automaton construction by including corner information, as suggested in example 5.10. The result is a weighted finite graph (whose cost functions will also be denoted cost and reward), in which it is possible to solve the (mean-)cost optimality problems [CLR90, Kar78, ZP96].

An important property of this graph is that, given a finite run $\varrho : (\ell_0, v_0) \to (\ell_1, v_1) \to \ldots \to (\ell_n, v_n)$ in $\mathcal{A}$, there exist two finite paths $\pi : (\ell_0, R_0, \alpha_0) \to (\ell_1, R_1, \alpha_1) \to \ldots \to (\ell_n, R_n, \alpha_n)$ and $\pi' : (\ell_0, R_0, \alpha'_0) \to (\ell_1, R_1, \alpha'_1) \to \ldots \to (\ell_n, R_n, \alpha'_n)$ in $\mathsf{CP}(\mathcal{A})$ such that $v_i \in R_i$ for every i, α_i and α'_i are corners

of R_i, and $\mathsf{cost}(\pi) \leqslant \mathsf{cost}(\varrho) \leqslant \mathsf{cost}(\pi')$. Conversely, for every finite path $\pi : (\ell_0, R_0, \alpha_0) \to (\ell_1, R_1, \alpha_1) \to \ldots \to (\ell_n, R_n, \alpha_n)$ in $\mathsf{CP}(\mathcal{A})$, for every $\varepsilon > 0$, we can construct a real run $\varrho : (\ell_0, v_0) \to (\ell_1, v_1) \to \ldots \to (\ell_n, v_n)$ in $\mathcal{A}$ such that for every index i, $v_i \in R_i$, and $|\mathsf{cost}(\varrho) - \mathsf{cost}(\pi)| < \varepsilon$.

There is thus a strong relation between finite runs in $\mathcal{A}$ and finite paths in $\mathsf{CP}(\mathcal{A})$. Computing the optimal cost for reaching a given goal in $\mathcal{A}$ reduces to computing the optimal cost for reaching a distinguished set of states in the discrete weighted graph $\mathsf{CP}(\mathcal{A})$.

The case of optimal mean-cost needs some more work, the corner-point abstraction can nonetheless be used to compute it. We first recall that in a finite weighted graph, the optimal mean-cost can be computed as the mean cost of a reachable (simple) cycle that minimizes that value [Kar78] — we call such a cycle an *optimal cycle*. Then, we prove that the mean-cost of an infinite run in $\mathcal{A}$ cannot be any better than the optimal cycle in $\mathsf{CP}(\mathcal{A})$. This can be proven by taking longer and longer prefixes of an infinite run ϱ, and at the limit, the ratio will always be larger than the mean-cost of the optimal cycle in $\mathsf{CP}(\mathcal{A})$. Write ϱ_n for the prefix of length n of ϱ. Applying the previous result on finite runs, we can construct a finite path π_n in $\mathsf{CP}(\mathcal{A})$ such that $\mathsf{cost}(\pi_n) \leqslant \mathsf{cost}(\varrho_n)$. We can decompose π_n into cycles as schematically depicted below:



The linear part of π_n is cycle-free, hence has a bounded length, and its costs will somehow become negligible when n tends to $+\infty$. The mean-cost of every cycle is no better than the optimal cycle of $\mathsf{CP}(\mathcal{A})$. Hence, at the limit, the mean-cost of ϱ will not be better than the mean-cost of the optimal cycle in $\mathsf{CP}(\mathcal{A})$. Conversely, paths in the corner-point abstraction can be approximated by real runs in the original automaton with costs and rewards that are very close to the one in the corner-point abstraction. The construction is presented in details in [BBL08].

The size of the corner-point abstraction is exponential in the size of the original automaton (a region R has at most $|X|$ corner-points, where X is the set of clocks of the automaton), *i.e.*, as is the size of the region automaton. Using non-determinism, we can guess optimal paths (respectively cycles) in $\mathsf{CP}(\mathcal{A})$, without first computing the full graph. This non-deterministic algorithm uses polynomial space, hence the PSPACE upper bound for the

two optimization problems. The PSPACE lower bound can be easily obtained by reduction to the reachability problem in timed automata, for appropriate cost functions.

5.3.3 Partial conclusion and related work

In this section, we have presented the decidability results for the two basic optimization problems on weighted timed automata. This is really encouraging because the theoretical complexity of these problems is not very high (in the context of real-time model-checking).

We have seen in chapter 3 that the region automaton construction is not implemented in tools like Uppaal [BDL+06] or Kronos [BDM+98] but a symbolic approach (based on zones) is preferred and implemented. Following this approach, a symbolic approach for computing optimal costs for reaching some distinguished set of goal locations, based on an extension of zones, called *priced zones*, has been developed in [LBB+01] and later implemented in the tool Uppaal-Cora.[3] The paper [BLR04] reports algorithms and applications of this tool. Also, optimal cost, given some constraint on another cost variable, is proven computable [LR05] and implemented in that tool. For the moment, the optimal mean-cost is still not implemented, because there is no data structure that has been developed to deal with that problem. This is however a very challenging (and non-trivial) line of research: there is no obvious extension of the priced zone symbolic representation which could be correct for the optimal mean-cost problem.

Recently, the corner-point abstraction has been used to prove the decidability of another optimization problem [FL08]. Along a run, the cost is now discounted with respect to the time which has elapsed. This extends the classical discounted payoff that we can find in the game theory literature [ZP96]. The optimal discounted cost is proven to be computable, and it uses the corner-point abstraction.

5.4 Optimal (reachability) timed games

We have seen the optimal cost and the optimal mean-cost were both computable in weighted timed automata in polynomial space. This is really encouraging to consider more involved problems. In this section, we consider the very similar problems, but no more in the context of closed systems, as in the previous section, but in the context of open systems. An open system

[3] `http://www.cs.aau.dk/~behrmann/cora/publications.html`

somehow models an interaction between the system itself and the environment it is embedded in. As often this is modelled as games [Thom02] and we will often use terminologies from game theory.

5.4.1 Weighted timed games

A *weighted timed game* is a weighted timed automaton in which transitions are decoupled into *controllable* transitions (played by the *controller*) and *uncontrollable* transitions (played by the *environment*).

Let $\mathcal{G} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathsf{cost})$ be a weighted timed game. We assume Goal locations are sink locations with cost 0 per time unit, and a loop on each of the locations with cost 0. A *(controller) strategy* in $\mathcal{G}$ from the initial state $s_0 = (\ell_0, \mathbf{0}_X)$ is a partial function f from $\mathsf{Runs}_{\mathsf{f}}(\mathcal{G}, s_0)$, the set of finite runs starting in s_0, into the set of controllable transitions of $\mathcal{G}$ plus the symbol λ (which is for 'delaying') such that:

- $f(s_0)$ is defined,
- if $f(\varrho)$ is defined and $\mathsf{last}(\varrho) = s$, then:
  - either $f(\varrho)$ is a controllable transition e in $\mathcal{G}$, and e is enabled from s, in which case $f(\varrho \xrightarrow{e})$[4] has to be defined, and for every uncontrollable transition u in $\mathcal{G}$ which is enabled from s, $f(\varrho \xrightarrow{u})$ has to be defined;
  - or $f(\varrho)$ is λ, and there exists $d > 0$ such that $f(\varrho \xrightarrow{d'})$ is defined for every $0 < d' \leqslant d$, and $f(\varrho \xrightarrow{d'}) = \lambda$ for every $< d' < d$, and for every uncontrollable transition u enabled at some $s + d'$ with $0 \leqslant d' \leqslant d$, $f(\varrho \xrightarrow{d',u})$ has to be defined.

This is the notion of strategy that has been introduced in [BCFL04], but in the literature, alternative definitions can be found, we mention for instance the one used in [ABM04, BBR05], where f is a partial function that associates to a run $\varrho \in \mathsf{Runs}_{\mathsf{f}}(\mathcal{G}, s_0)$ a pair $(d, e) \in \mathbb{R}_{\geqslant 0} \times E$ which describes the next move to be done (wait d time units, and take edge e) from the current configuration. These different definitions yield different properties, we will mention one of them later.

A strategy f is said *memoryless* if for all runs $\varrho, \varrho' \in \mathsf{Runs}_{\mathsf{f}}(\mathcal{G}, s_0)$, $\mathsf{last}(\varrho) = \mathsf{last}(\varrho')$ implies $f(\varrho) = f(\varrho')$. Memoryless strategies are somehow 'simple' strategies that do not take past into account to make the next decision. A strategy f gives rise in a natural way to a set of *maximal plays* (because

[4]The notation $\varrho \xrightarrow{e}$ is a shortcut for the run ϱ extended by transition e.

$\mathcal{G}$ is supposed to be non-blocking, they correspond to infinite runs generated by the strategy) denoted $\mathsf{plays}_{\mathcal{G}}(f)$. The strategy f is *winning* (for the reachability goal) if and only if all (maximal) plays of $\mathsf{plays}_{\mathcal{G}}(f)$ end up in Goal.

The classical reachability game problem asks, given a timed game $\mathcal{G}$, whether there is a winning controller strategy for the reachability goal. Classical reachability games have been considered in the context of timed systems in the 90's, and deciding those games is EXPTIME-complete [AMPS98, HK99]. For those games, memoryless strategies are sufficient, and they are even region-invariant if we take the notion of strategy in [ABM04, BBR05], but they are not region-invariant with the notion of strategy we have chosen here, see [BCFL04].

With weighted timed games, an optimality criterion can be added to those games. The cost of a winning strategy f is defined as:

$$\mathsf{cost}_{\mathcal{G}}(f) = \sup\{\mathsf{cost}(\varrho) \mid \varrho \in \mathsf{plays}_{\mathcal{G}}(f)\}$$

Note that if f is a winning strategy, then for every $\varrho \in \mathsf{plays}_{\mathcal{G}}(f)$, $\mathsf{cost}(\varrho) < +\infty$. However it might be the case that $\mathsf{cost}_{\mathcal{G}}(f) = +\infty$.

The aim of the controller is to optimize this value and we want to compute the optimal cost the controller can ensure, whatever the environment does, which can be formally written as:

$$\mathit{opt\_cost}_{\mathcal{G}} = \inf\{\mathsf{cost}_{\mathcal{G}}(f) \mid f \text{ winning strategy}\}$$

We will consider the following decision problem which asks, given a threshold $c \in \mathbb{Q}_{\geqslant 0}$, whether there is some strategy f such that $\mathsf{cost}_{\mathcal{G}}(f) \leqslant c$. We will call this problem the *bounded cost problem*. We will also be interested in synthesizing *almost-optimal* strategies, that is for every $\varepsilon > 0$, computing a strategy f_ε which is *ε-optimal*: $\mathit{opt\_cost}_{\mathcal{G}} \leqslant \mathsf{cost}_{\mathcal{G}}(f) \leqslant \mathit{opt\_cost}_{\mathcal{G}} + \varepsilon$.

***Example* 5.11 (Taken from [BCFL04])** We consider the weighted timed automaton of example 5.2 (page 75). Dashed (respectively plain) arrows are now for uncontrollable (respectively controllable) transitions. Depending on the choice of the environment (going to location ℓ_2 or ℓ_3), the accumulated cost along plays of the game is either $5t + 10(2-t) + 1$ (through ℓ_2) or $5t + (2-t) + 7$ (through ℓ_3) where t is the delay elapsed in location ℓ_0. The optimal cost the controller can ensure is thus $\inf_{t \leqslant 2} \max(5t + 10(2-t) + 1, 5t + (2-t) + 7) = 14 + \frac{1}{3}$, and the optimal time for firing transition e_1 is when $t = \frac{4}{3}$. The optimal strategy for the controller is thus to wait in location ℓ_0 until $x = \frac{4}{3}$, and then enter location ℓ_1. Then, the environment chooses to go either to ℓ_2 or to ℓ_3, and finally when the value of x reaches 2, the controller goes to the goal location ☺. ⌟

***Remark* 5.12** Let us mention that in the above example, the optimal cost is non-integral, contrary to the case of closed systems. This means in particular that no region-based technology (and even corner-point abstraction) can be used to solve optimal timed games. ⌟

5.4.2 Discussion

In the late 90's, optimal-time timed games (*i.e.*, weighted timed games where cost represents time elapsing) have been considered [AM99], and the complexity has been made precise rather recently [JT07] using strategy improvement techniques.

Theorem 5.13 ([AM99, JT07]) *Optimal-time in reachability timed games is computable. It is* EXPTIME*-complete.*

The reason is that the region abstraction needs not be refined to compute the optimal time.

***Remark* 5.14** Note also that the EXPTIME upper bound could have been computed as follows: solve the reachability game classically, and record the corresponding memoryless winning strategy (using for instance a backward algorithm *à la* [AMPS98]), compute the maximal time τ for winning following that memoryless strategy (this needs to be bounded, otherwise it would not be winning), and then add an extra clock z which is never reset but is used in a guard $z \leqslant c$ (for c chosen non-deterministically not larger than τ) which constrains every transition leading to a location in Goal. The optimal time is the smallest c for which the transformed game is winning (because thanks to [AM99] we know that the optimal time is an integer). Finally as the value of τ is at most exponential (because the selected winning strategy is memoryless), this global algorithm only requires exponential time. ⌟

Then, in [LMM02], optimal timed games (with general costs) are considered, and a doubly-exponential time algorithm is designed for computing optimal cost (and synthesizing (almost-)optimal strategies) in *acyclic* timed games. The algorithm somehow extends classical min/max-algorithms for discrete games to timed games. Optimal timed games have further been studied from 2004 on.

In [ABM04], the 2EXPTIME upper bound mentioned above is improved to an EXPTIME upper bound. Note that this algorithm computes for every winning state the optimal cost for winning and provides a (possibly almost) optimal winning strategy. The algorithm which is proposed splits the state-space into polyhedra on which (roughly) optimal winning strategies are uniform, it is pretty involved, and relies on nice geometrical properties of the

state-space. Moreover, a family of weighted timed games is given, for which it is unavoidable to split the set of winning states into an exponential number of pieces.

The work done in [BCFL04] has considered properties of optimal winning strategies. In particular, it is shown that optimal winning strategies may need some memory to win, but also that not much memory is required (the accumulated cost since the beginning of the play is sufficient). A semi-algorithm is proposed, which, if it converges, compute the set of winning states, together with the optimal cost for winning. We have implemented [BCFL05] the semi-algorithm, based on hybrid games, on top of the tool HyTech [HHWT97]. However, only a quite strong hypothesis ensures the termination of the semi-algorithm (some strongly non-Zenoness of the cost function).

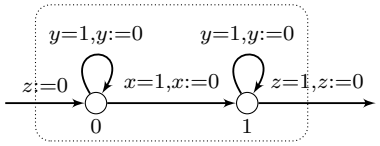
5.4.3 Undecidability results

The first undecidability result has come as a surprise from colleagues in Belgium [BBR05], it requires weighted timed games with five clocks or more. We have proposed an improved undecidability proof [BBM06], which only uses weighted timed games with three clocks.

☞ **Theorem 5.15 ([BBR05, BBM06])** *The bounded cost problem for weighted timed games with no less than three clocks is undecidable.*

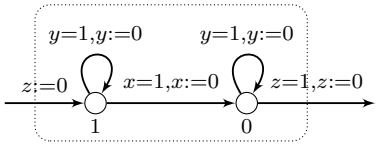
***Remark* 5.16** This result does not formally imply the non-computability of the optimal cost. Furthermore we did not manage to turn the proof in such a way that the optimal cost be non-computable. However, this is already bad news. ⌟

We will present the basic ideas of the undecidability proof proposed in [BBM06], which we think is quite instructive. First we consider the two small modules that are depicted below. The module $\mathsf{Add}_z^{+x}(x,y)$ (respectively $\mathsf{Add}_z^{+(1-x)}(x,y)$) uses z as an extra clock, lets the values of x and y at the end of the module be the same as at the beginning of the module, increases the cost by x_0 (respectively $1-x_0$) if x_0 is the value of x when entering the module.



The cost is increased by x_0
The values of x and y are unchanged

Module $\mathsf{Add}_z^{+x}(x,y)$



The cost is increased by $1-x_0$
The values of x and y are unchanged

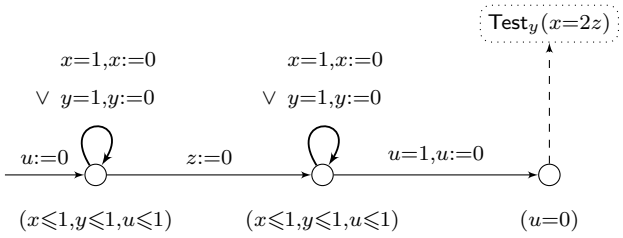
Module $\mathsf{Add}_z^{+(1-x)}(x,y)$

Concatenating these modules, one can implement various cost functions (non-negative linear combinations of x_0, y_0, $1-x_0$, $1-y_0$). In particular, we can implement the two cost functions cost_1 and cost_2 defined as follows:

$$\mathsf{cost}_1(x_0, y_0) = 2x_0 + (1 - y_0) + 2 \qquad \mathsf{cost}_2(x_0, y_0) = 2(1 - x_0) + y_0 + 1$$

Now, it is easy to check that $2x_0 > y_0$ implies $\mathsf{cost}_1(x_0, y_0) > 3$, whereas $2x_0 < y_0$ implies $\mathsf{cost}_2(x_0, y_0) > 3$. Moreover, if $2x_0 = y_0$, then $\mathsf{cost}_1(x_0, y_0) = \mathsf{cost}_2(x_0, y_0) = 3$. Hence if we are in a state with $x = x_0$ and $y = y_0$, and if the choice of the cost function is given to the environment, it can enforce a cost (strictly) larger than 3 if and only if $2x_0 \neq y_0$. Otherwise, the cost will be 3, whatever is the choice of the environment. This will later serve as a module to check whether twice the value of x is equal to the value of y. We denote this test module $\mathsf{Test}_z(2x = y)$, with the subscript z to indicate that an extra clock z is used in the module.

To simulate a two-counter machine, the idea is to store the value of a counter c into a clock, whose value will be, at distinguished points in time, $\frac{1}{2^c}$. Hence, to store the values of two counters, one needs two clocks. Assume an instruction increments the first counter, and lets the second counter unchanged. Assume furthermore that the value of the first counter is c and stored in clock x, whereas the value of the second counter is d and is stored in clock y. We consider the module depicted on the figure below, which will simulate the above instruction (the value of the first counter is initially stored in clock x and finally in clock z).



$$\begin{pmatrix} x{=}\frac{1}{2^c} \\ y{=}\frac{1}{2^d} \\ z{=}\star \end{pmatrix} \qquad \begin{pmatrix} x{=}\frac{1}{2^c} \\ y{=}\frac{1}{2^d} \\ z{=}\alpha \end{pmatrix}$$

The duration of an execution in that module is one time unit (condition checked by the extra clock u). It is not difficult to check that the final values for x and y correspond to their initial values. The final value for z has been non-deterministically guessed during the execution, so can be anything within the interval $[0, 1]$. An uncontrollable transition leads to the

test module $\mathsf{Test}_y(x = 2z)$ that we have described earlier. If (and only if) the guess for z has been correct (or equivalently $2\alpha = \frac{1}{2^c}$) the environment has no strategy to get a cost value larger than 3. There is no cost labelling locations of the main automaton, we only add a discrete cost of +3 when reaching the halting state. In that reduction,

the two-counter machine halts if, and only if,
the controller has a winning strategy with cost no more than 3
in the weighted timed game

It is worth noticing that the described reduction uses four clocks, and not three, as claimed. However, we can get rid of clock u using the following trick: the value of the second counter d is now stored by the value $\frac{1}{3^d}$ (note that the choice of $\frac{1}{2^c}$ and $\frac{1}{3^d}$ is arbitrary, it could be $\frac{1}{p^c}$ and $\frac{1}{q^d}$ for p and q relatively prime integers). Indeed we can prove that the constraint $u = 1$ at the end of the module can be replaced by the constraints that the value of x is a negative power of 2 and the value of y is a negative power of 3. Testing that the value of x is a negative power of 2 can be done by iteratively multiplying the value of x by 2 (done using the $\mathsf{Test}_y(z = 2x)$ module) and eventually reaching 1. Finally the constraint that the last location of the module be transient is done by adding a positive cost to that location, and requiring the controller to have a strategy with cost no more than 3.

5.4.4 Decidability results

The previous undecidability result is rather bad news, but more positive results have been looked at, though.

Theorem 5.17 ([BBR05]) *The optimal cost in single-clock weighted timed games is computable if we restrict to a stopwatch cost.*

In the restricted case mentioned in the above theorem, the semi-algorithm proposed in [BCFL04] terminates, because roughly, classical regions never need to be split and are thus correct.

More recently, optimal cost in weighted timed games with one clock (but arbitrary cost) has been proven computable [BLMR06] (though in a restricted *turn-based* framework where locations are either controllable — *i.e.* all transitions leaving this location are controllable — or uncontrollable). This work is joint work with Kim G. Larsen, Nicolas Markey, and Jacob I. Rasmussen.

☞ **Theorem 5.18 ([BLMR06])** *The optimal cost in turn-based single-clock weighted timed games is computable. Furthermore, for every $\varepsilon > 0$, we can compute ε-optimal and* ***memoryless*** *strategies.*

***Remark* 5.19** The complexity of the algorithm we have designed (which relies on the fix-point algorithm proposed in [BCFL04]) is 3EXPTIME, and the best lower bound that is known is PTIME-hard. ⌟

5.4.5 Partial conclusion and remarks

In this section, we have presented the problem of optimal timed games, where the aim of the controller is to optimize the cost for reaching some designated set of goal locations, whatever the environment does. The general problem is unfortunately undecidable, and only restricted classes of systems yield decidability.

It is worth noticing here that the undecidability proof presented in subsection 5.4.3 can easily be adapted to optimal mean-cost timed games (extensions of mean-payoff games), where the goal for the controller is to optimize the mean-cost that can be achieved. The idea is to add a transition from the Goal location to the initial location, let the reward be 1 when a Goal location is visited. The question is then whether it is possible to have the mean-cost be no more than 3 or not. And in the reduction, this is equivalent to testing that the two-counter machine halts. Natural questions can be whether one can find cost/reward functions that would yield a decidable optimal game problem. Very recently, a restricted class of mean-cost timed games has been proven decidable [JT08].

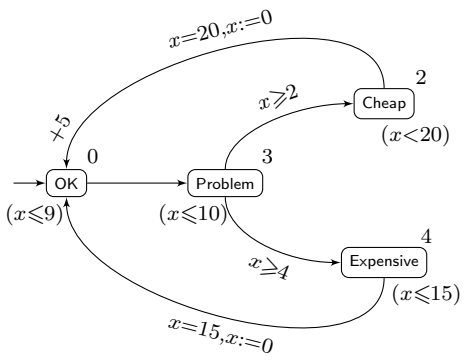
5.5 Model-checking problems

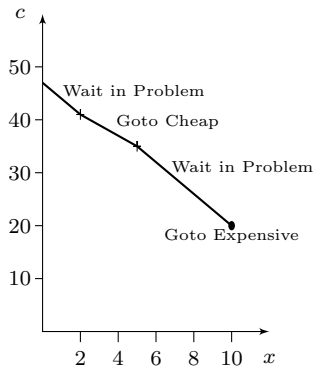
In this section, we will consider another kind of more involved problems, and will study model-checking problems where specifications are given by extensions of classical temporal logics like LTL [Pnu77] or CTL [CE82] with cost constraints. In chapter 4, we have already studied extensions of LTL with quantitative constraints put on delays between events. Now, the quantitative constraints are no more relative to delays, but rather to the cost function. We will consider two possible logics, one based on CTL, and the other one based on LTL, and we will give decidability and undecidability results, that will unfortunately also be rather bad news.

Before going to the definition of the logics, we give an example of systems that will help illustrate our logics.

***Example* 5.20** The weighted timed automaton depicted below models a never-ending process of repairing problems, which are bound to occur repeatedly with a certain frequency. The repair of a problem has a certain cost, captured in the

model by a cost variable (rates are indicated close to locations, and discrete costs are indicated at the end of the edge). As soon as a problem occurs (modelled by the Problem location) the value of the cost grows with rate 3, until actual repair is taking place in one of the locations Cheap (rate 2) or Expensive (rate 4). At most 20 time units after the occurrence of a problem it will have been repaired one way or another.





In this setting we are interested in properties concerning the cost of repairs. For instance, we would like to express that whenever a problem occurs, it *may* be repaired (*i.e.* reach the location OK) within a total cost of 47. The figure to the right above gives the minimum cost of repair —as well as an optimal strategy— for any configuration of the form $(\mathsf{Problem}, x)$ with $x \in [0, 10]$. Correspondingly, the minimum cost for reaching OK from configurations of the form (Cheap, x) (respectively $(\mathsf{Expensive}, x)$) is given by the expression $45-2x$ (respectively $60-4x$). Symmetrically, we would like to express properties on the worst cost to repair, or to link the uptime with the (best, worst) cost of repairing. As will be illustrated later, extending temporal logics with cost information provides a nice setting for expressing such properties.

⌟

5.5.1 WCTL: an extension of CTL with cost constraints

The logic WCTL [5] is a branching-time logic which extends CTL with cost constraints on modalities. It has been first defined in [BBR04]. The syntax of WCTL [6] over the finite set of atomic propositions AP is given by the following grammar:

$$\mathsf{WCTL} \ni \varphi \ ::= \ p \ \mid \ \varphi \vee \varphi \ \mid \ \neg\varphi \ \mid \ \mathbf{E}\,(\varphi\,\mathbf{U}_{\sim c}\,\varphi) \ \mid \ \mathbf{A}\,(\varphi\,\mathbf{U}_{\sim c}\,\varphi)$$

where $p \in \mathsf{AP}$, $c \in \mathbb{Q}_{\geqslant 0}$ and $\sim \in \{<, \leqslant, =, \geqslant, >\}$.

[5] WCTL stands for "Weighted CTL".

[6] The original syntax given in [BBR04] allowed more general constraints on modalities, but we restrict to that fragment to get some decidability results.

Formulas of WCTL are interpreted over configurations of a weighted timed automaton $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L}, \mathsf{cost})$. Given s such a configuration of $\mathcal{A}$ and φ, the satisfaction relation is defined inductively as follows (atomic propositions and Boolean combinations are omitted because they are straightforward), writing $s \models \varphi$ whenever φ is satisfied from s:

$$\begin{array}{rcl} s \models \mathbf{E}\,(\varphi\,\mathbf{U}_{\sim c}\,\psi) & \Leftrightarrow & \text{there exists } \varrho \in \mathsf{Runs}(\mathcal{A}, s) \text{ and a position } \varpi \\ & & \text{along } \varrho \text{ such that } \mathsf{cost}(\varrho_{[\varpi_0;\varpi]}) \sim c,\ \varrho_{[\varpi]} \models \psi, \text{ and} \\ & & \text{for every position } \varpi_0 < \varpi' < \varpi \text{ along } \varrho,\ \varrho_{[\varpi']} \models \varphi \\ s \models \mathbf{A}\,(\varphi\,\mathbf{U}_{\sim c}\,\psi) & \Leftrightarrow & \text{for every } \varrho \in \mathsf{Runs}(\mathcal{A}, s), \text{ there is a position } \varpi \\ & & \text{along } \varrho \text{ such that } \mathsf{cost}(\varrho_{[\varpi_0;\varpi]}) \sim c,\ \varrho_{[\varpi]} \models \psi, \text{ and} \\ & & \text{for every position } \varpi_0 < \varpi' < \varpi \text{ along } \varrho,\ \varrho_{[\varpi']} \models \varphi \end{array}$$

where ϖ_0 still denotes the initial position of the run, and where we take the notations of subsection 2.2.2. As for CTL, there might be two possible interpretations, depending on the meaning of the term 'position'. In the following, this will not have much influence on the results we get, hence do not insist anymore on that distinction.

***Example* 5.21** The weighted timed automaton $\mathcal{A}$ of example 5.2 (page 75) satisfies the property $s_0 \models \mathbf{E}\,\mathbf{F}_{\leqslant 10}\,\☺$, where $s_0 = (\ell_0, \mathbf{0}_X)$, because there is a run from s_0 that reaches ☺ and whose cost is no more than 10 (*eg.* the run ϱ given in example 5.2). ⌟

***Remark* 5.22** In the (simplified) definitions we gave for weighted timed automata and WCTL, only a single cost variable is assumed to measure quantities in the system. However, as said in remark 5.3, there could be several cost variables in the weighted timed automata, and each modality could be decorated by a constraint involving one of the cost variables. For instance, there is a natural cost variable that measures time elapsing (rate 1 in every location, and cost 0 for every edge), that we will next denote time. ⌟

***Example* 5.23** We go back to example 5.20 (page 89). It is always possible to repair a problem with cost at most 47 can be written in WCTL with the following formula:

$$\mathbf{A}\,\mathbf{G}\,\big(\mathsf{Problem} \Rightarrow\ \mathbf{E}\,\mathbf{F}_{\leqslant 47}\,\mathsf{OK}\big).$$

We can also express that the worst cost to repair is 56, in the sense that state OK can always be reached within this cost:

$$\mathbf{A}\,\mathbf{G}\,\big(\mathsf{Problem} \Rightarrow\ \mathbf{A}\,\mathbf{F}_{\leqslant 56}\,\mathsf{OK}\big).$$

Now, considering time as a special case of a cost, we can express properties relating the time elapsed in the OK state and the cost to repair:

$$\mathbf{A}\,\mathbf{G}\,\big(\neg\,\mathbf{E}\,(\mathsf{OK}\ \ \mathbf{U}_{\mathsf{time}\geqslant 8}\,(\mathsf{Problem} \wedge \neg\,\mathbf{E}\,\mathbf{F}_{\mathsf{cost}<30}\,\mathsf{OK}))\big).$$

This expresses that if the system spends at least 8 (consecutive) time units in the OK state, then the next Problem can be repaired with cost at most 30. ⌟

Note that when there is a single cost variable which corresponds to time elapsing, WCTL coincides with the classical logic TCTL [ACD93], whose model-checking is known to be PSPACE-complete (even under a single-clock assumption in the timed automaton [LMS04]). However, model-checking WCTL will turn out not to be as easy as model-checking TCTL. Indeed, it was first proven in [BBR04] that for five clocks or more, the model-checking of WCTL is undecidable. We have then further proven that already three clocks lead to undecidability [BBM06].

☞ **Theorem 5.24 ([BBR04, BBM06])** *Model-checking* WCTL *is undecidable for weighted timed automata with three clocks or more.*

On the other hand, restricting to single-clock weighted timed automata yields a surprising improvement for the model-checking of WCTL [BLM07, BLM08]. Indeed, this becomes no more difficult than model-checking TCTL!

☞ **Theorem 5.25 ([BLM07, BLM08])** *Model-checking* WCTL *is* PSPACE*-complete for single-clock weighted timed automata (with possibly several cost variables).*

***Remark* 5.26** Note that the above result assumes the *dense-time* semantics, *i.e.* the time domain be $\mathbb{Q}_{\geqslant 0}$ or $\mathbb{R}_{\geqslant 0}$. Indeed, it has been proven in [BBR04] that if we restrict to the discrete-time domain $\mathbb{N}$, the model-checking of WCTL becomes decidable (the time space can be discretized). ⌟

We will not take much time explaining the undecidability result. Basically a reduction similar to the one done for weighted timed games can be used (roughly, when the second player is used, we can use the $\mathbf{A}$-quantification instead).

On the other hand, the decidability proof is rather technical, and uses the following ideas: for each sub-formula (in a bottom-up manner, hence starting from atomic propositions), we compute a sufficient granularity that defines regions within which the truth of that sub-formula is uniform; Then, each such region is labelled by the truth of the current sub-formula, and the algorithm goes on with higher sub-formulas. It is thus interesting to notice that there may be different cost variables constraining different modalities. Globally, we can prove that an exponentially small granularity (in the size of the formulas, and in the constants appearing in the formula) needs to be distinguished, yielding immediately an exponential time algorithm to solve the model-checking problem. A highly non-deterministic algorithm can be

used instead, as it was done in [HKV96] for TCTL (though we were not aware of that paper when designing our polynomial space algorithm for WCTL).

5.5.2 WMTL: an extension of LTL with cost constraints

The logic WMTL [7] is a linear-time logic which extends LTL with cost constraints on modalities. It has been introduced in [BM07]. The syntax of WMTL over AP is given by the following grammar:

$$\mathsf{WMTL} \ni \varphi \ ::= \ p \ \mid \ \varphi \vee \varphi \ \mid \ \neg\varphi \ \mid \ \varphi \, \mathbf{U}_{\sim c} \, \varphi$$

where $p \in \mathsf{AP}$, $c \in \mathbb{Q}_{\geqslant 0}$ and $\sim \in \{<, \leqslant, =, \geqslant, >\}$.

As for MTL (see chapter 4), formulas of WMTL are interpreted over (finite or infinite) runs from some position along those runs. Let ϱ be a (finite or infinite) run in $\mathcal{A}$, and ϖ be a position along ϱ. The satisfaction relation is defined as follows (atomic propositions and Boolean combinations are omitted):

$$\begin{array}{rcl} (\varrho, \varpi) \models \varphi \, \mathbf{U}_{\sim c} \, \psi & \Leftrightarrow & \text{there exists a position } \varpi' > \varpi \text{ along } \varrho \text{ such that} \\ & & \mathsf{cost}(\varrho_{[\varpi;\varpi']}) \in I, \ (\varrho, \varpi') \models \psi, \\ & & \text{and for every position } \varpi < \varpi'' < \varpi', \ (\varrho, \varpi'') \models \varphi \end{array}$$

where we stick to the notations of subsection 2.2.2. As now usual, there might be two possible interpretations for this logic, depending on the meaning of the term 'position'. In the following, we will focus on the simplest interpretation, *i.e.* the $\models_{\mathsf{f}}^{\mathsf{point}}$-semantics. Indeed, WMTL extends MTL, and all other semantics already yield undecidable model-checking problems for MTL (recall section 4.4).

***Remark* 5.27** As for WCTL, we can mix cost variables in the formulas. In the following, we will allow such features, and give (un)decidability results, distinguishing between the number of cost variables that are used. ⌟

***Example* 5.28** Back to our example 5.20 (page 89), we can express that there is no path from OK back to itself in time less than 10 *and* cost less than 20. This is achieved by showing that no path satisfies the following formula:

$$\mathsf{OK} \ \mathbf{U} \ (\mathsf{Problem} \wedge (\neg\mathsf{OK}) \ \mathbf{U}_{\mathsf{time} \leqslant 10} \ \mathsf{OK} \wedge (\neg\mathsf{OK}) \ \mathbf{U}_{\mathsf{cost} \leqslant 20} \ \mathsf{OK}).$$

⌟

[7] WMTL stands for "Weighted MTL".

☞ **Theorem 5.29 ([BM07, BLM08])** *Model-checking* WMTL *is decidable for the* $\models_f^{point}$*-semantics, when we restrict to single-clock weighted timed automata with a single stopwatch cost variable. Any extension leads to the undecidability of the model-checking problem. This can be summarized in the following table (in bold face we indicate results that are proven, corollaries are written in non-bold face):*

	stopwatch costs			normal costs	
	1 variable	2 variables	3 variables	1 variable	2 variables
0 clock	*decidable*	*?*	***undecidable***	*?*	***undecidable***
1 clock	***decidable***	***undecidable***	*undecidable*	***undecidable***	*undecidable*
2 clocks	***undecidable***	*undecidable*	*undecidable*	*undecidable*	*undecidable*

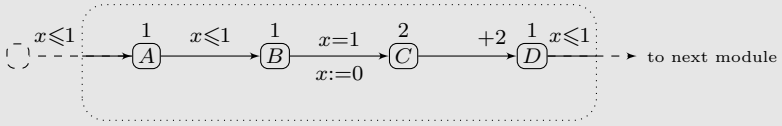
The decidability result relies on a construction based on stopwatch single-clock alternating timed automata, that extends the construction made in section 4.4.2 (the single-clock hypothesis appears rather quickly when trying to extend the proof).

The undecidability proof relies on ideas similar to the one done for optimal reachability timed games, except that one needs to store the values of the two counters in a single clock. We do so by storing the value $\frac{1}{2^c 3^d}$ in the unique clock, where c (respectively d) is the value of the first (respectively second) counter. All operations have to be done in a more clever way than it was done for optimal timed games, we give parts of the reduction in table 5.3, the complete reduction is given in [BM07, BLM08].

5.6 Conclusion and further works

Timed automata extended with cost information have been extensively studied in the past few years. We have presented here some of the results which have been obtained in the context of model-checking and games. Basic optimization problems are surprisingly easy to do, but extensions to games or to more general model-checking problems are hard in general. We would like to point out the new interest in single timed automata that arises from this study. Indeed, some of the problems become decidable in that case (with a rather reasonable complexity). Restriction to single clock models already yielded a dramatic improvement in the complexity of the verification of reachability properties in timed automata (it becomes NP-complete, see [LMS04]), and yielded decidability to alternating timed automata [LW05, LW08]. Though that might be seen as a drastic restriction, this is however rather interesting because it allows to model systems with (simple) timing constraints, but with other quantitative aspects.

The following module, together with some WMTL formula, will simulate an increment of the first counter.



The WMTL formula needs to express that the value of the clock be divided by 2 (to safely implement the increment of the first counter). This can be achieved by expressing that no time elapses in locations A and D, and that the global cost in that module accumulates to 3. It can be written:

$$\mathbf{G}\,(A \rightarrow (A\,\mathbf{U}_{=0}\,\neg A)) \wedge \mathbf{G}\,(D \rightarrow (D\,\mathbf{U}_{=0}\,\neg D)) \wedge \mathbf{G}\,(A \rightarrow (\neg D\,\mathbf{U}_{=3}\,D))$$

The following (more involved) module simulates a test to zero and a decrement of the first counter. We let the reader imagine the WMTL that could help simulating the instruction.

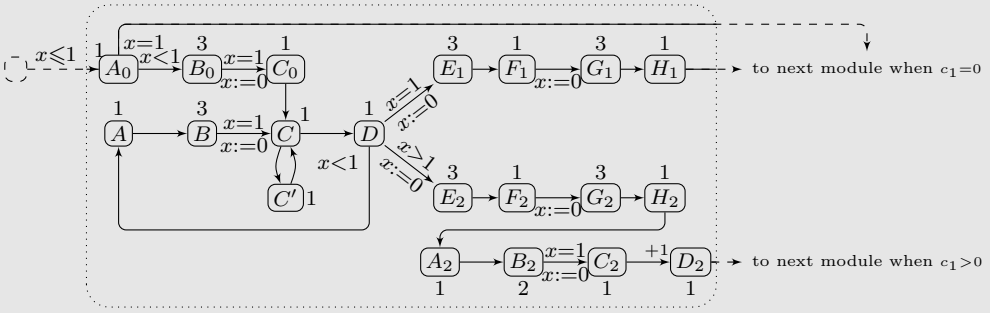


Table 5.3: Undecidability of WMTL: some ideas

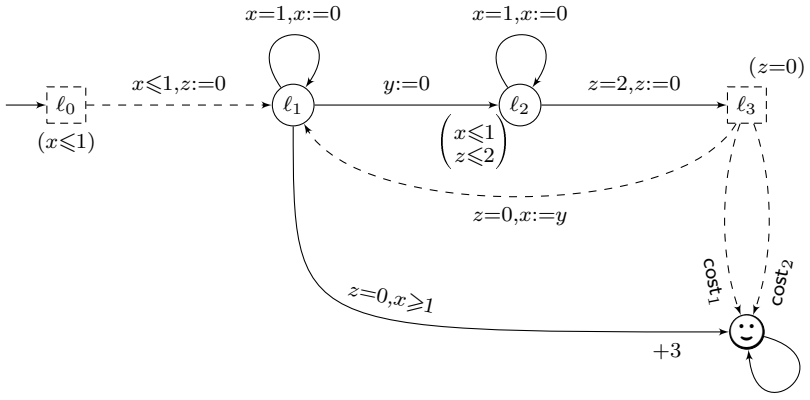
We have several works in progress on the model of weighted timed automata, we will describe two of them. Some other possibilities include the application of strategy improvement techniques to get further decidability results. Indeed, we have recently applied these techniques to prove the computability of the optimal cost in weighted games, where the underlying model is not a timed automaton, but a strong-reset hybrid automaton [BBJ+08]. Similar techniques have been used to compute optimal time and optimal mean-cost in (restricted classes of) timed games [JT07, JT08].

Approximate the optimal cost in timed games. Formally, we have not proven that the optimal cost was not computable in weighted timed

games, but rather that if it was computable, it would have a complicated form (because deciding whether it is below a rational threshold is undecidable). An interesting research direction is then to try to compute approximate optimal costs instead (in practice, that would be sufficient, as soon as we can bound the distance to the optimal value). Indeed, the study we have done so far does not imply that this problem should be complex.

A natural idea to get approximate values is to use the iterative semi-algorithm designed in [BCFL04]. Indeed, this computation, if it does terminate, computes the optimal cost of the game. However, we have not managed yet to prove that the values that are computed converge to the optimal cost in the general case (when the algorithm does not terminate). Furthermore, we do not quite see how to bound the distance to the optimal cost. We have thus tried to design another (semi-)algorithm, that would compute an approximation of the optimal cost from below, but all our attempts so far have failed (for instance, discretizing and refining the granularity of the discretization is not correct). This is however a challenging issue, that we should continue working on. I have started working on that subject with several people including Nicolas Markey, Joël Ouaknine, Jean-François Raskin, and James Worrell.

We now would like to illustrate with an example the kind of results we would like to obtain. We consider the following weighted timed automaton, where we use the macro $x := y$ that assigns the value of y to clock x (we know that this macro can be removed [BDFP04]). Furthermore, we have seen in page 87 that we can implement many cost functions, for instance $\mathsf{cost}_1 = 3 - 2x + y$ and $\mathsf{cost}_2 = 3 + 2x - y$. We will use these macros in the example. This game is turn-based (dashed edges are for the environment, whereas plain edges are for the controller; ℓ_0 and ℓ_3 are thus uncontrollable locations). The Goal location is ☺.



The algorithm designed in [BCFL04] does not terminate on that example. In this game, the environment first chooses a time at which he fires the first transition, and then the play will loop for some time, until one of the players decides to reach the goal location ☺. In a loop, the controller has a single choice, which is the delay in ℓ_1, all other choices are determined by the clock constraints.

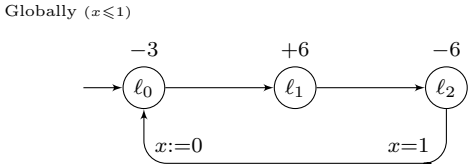
The optimal cost of that game for the controller is 3, and the controller has an optimal winning strategy, which consists in delaying in ℓ_1 a delay d such that $2 - d$ is equal to twice the value of x when entering ℓ_1. That way, when entering state ℓ_3, the value of y is twice the value of x, and thus the environment cannot make the controller pay more than 3 (both edges leading to the winning location ☺ will then have cost 3). On the other hand, if the controller does not follow this strategy, the environment can reach the goal location ☺ with a cost strictly larger than 3. Now, if the controller keeps playing that strategy, the value of clock x will eventually satisfy $x \leqslant 1$ when entering state ℓ_1, and thus he can escape to the goal location ☺, and pay a cost of 3. Note that this optimal winning strategy is memoryless, but it generates plays of unbounded length.

Let us consider now almost-optimal winning strategies. Fix an $\varepsilon > 0$. We will see that the controller has a winning strategy which is ε-optimal and for which the length of the prefix (up to ☺) of all plays have length bounded by $\mathcal{O}\Big(-\log_2(\varepsilon)\Big)$. Choose n integer such that $\frac{1}{2^{n-1}} < \varepsilon$. Then the strategy just changes the first time location ℓ_1 is visited. If the value of x is smaller than $\frac{1}{2^n}$, then take the next transition so that the value of y when entering state ℓ_3 be $\frac{1}{2^{n-1}}$. The controller makes a mistake in doing that (he does not multiply correctly by 2), but it will not cost that much, because it will cost no more than $3 + \frac{1}{2^{n-1}}$, which is smaller than $3 + \varepsilon$. The strategy for the next loops is the optimal one (the value of x is now large enough). Following that strategy, any play will be not longer than $k_1 + k_2 \cdot n$ (for some fixed value k_1 and k_2), and thus $k_1 + k_2 \cdot n$ iterations of the semi-algorithm of [BCFL04] will for sure give an ε-optimal strategy.

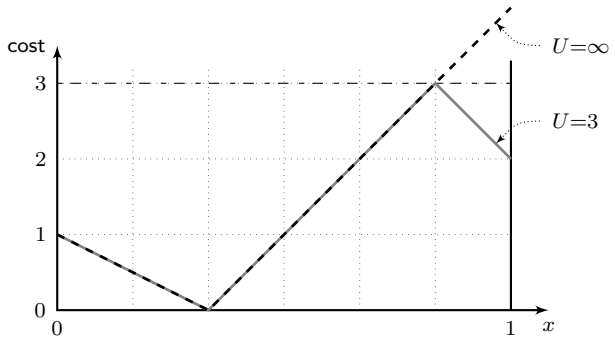
More accurate energy model. The model of weighted timed automata that we have studied in this chapter assumes non-negative costs labelling locations and edges. For some modelling problems, this may not be very accurate, because resources (or energy) can be consumed but also regained: we can think of autonomous robots that are equipped with solar cells for energy-harvesting, or also of batteries of laptops that may charge or discharge, depending on whether it is plugged or not. We thus extend the original model by allowing costs that can be negative or positive. Main challenges are now

to synthesize schedules or strategies that will ensure indefinite safe operation with the additional guarantee that energy will always be available, yet never exceeds a possible maximum storage capacity.

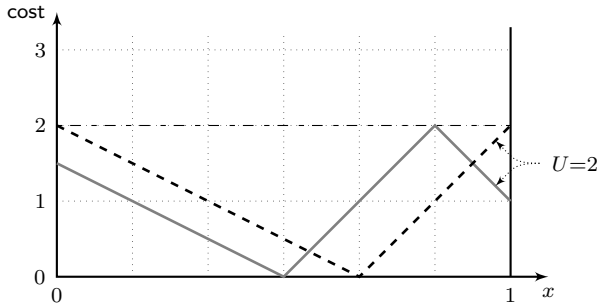
As a basic example, consider the weighted timed automaton below.



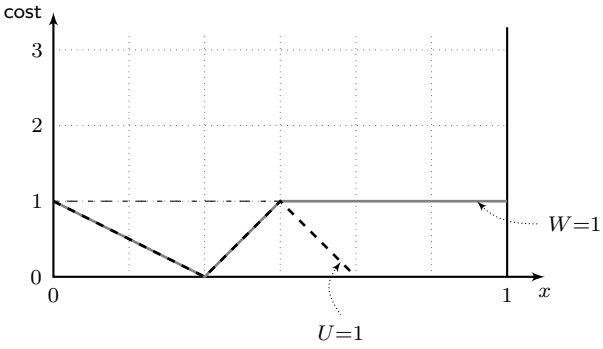
The infinite runs in this automaton repeatedly delay in ℓ_0, ℓ_1 and ℓ_2 for a total of precisely one time unit. The negative weights (-3 and -6) in ℓ_0 and ℓ_2 indicate the rates by which energy will be consumed, and the positive rate ($+6$) in ℓ_1 indicates the rate by which energy will be produced. Thus, for a given iteration the effect on the amount of energy remaining will depend highly on the distribution of the one time unit over the three locations. Let us observe the effect of lower and upper constraints on the energy level on so-called bang-bang strategies, where the behaviour remains in a given location as long as permitted by the given bounds. The next picture shows the bang-bang strategy given an initial energy-level of 1 with no upper bound (dashed line, $U = \infty$) or 3 as upper bound (solid line, $U = 3$). In both cases, it may be seen that the bang-bang strategy yields an infinite behaviour.



In the next picture, we consider the upper bound 2 ($U = 2$). We see that the bang-bang strategy reduces an initial energy level of $1\frac{1}{2}$ to 1 (solid line), and yet another iteration will reduce the remaining energy-level to 0. In fact, the bang-bang strategy—and it may be argued, any other strategy—fails to maintain an infinite behaviour for any initial energy-level *except for* 2 (dashed line).



In the next picture, we illustrate the case of an upper bound of 1. We see that the bang-bang strategy—and any other strategy—fails to complete even one iteration (dashed line, $U = 1$). We then propose an alternative *weak* notion of upper bounds, which does not prevent energy increasing behaviour from proceeding once the upper bound is reached but merely maintains the energy level at the upper bound. In this case, as also illustrated in the picture below (solid line, $W = 1$), the bang-bang strategy is quite adequate for yielding an infinite behaviour.



We have done some preliminary investigations in that direction [BFL$^+$08], distinguishing between several problems: (*i*) the game problem, existence of a strategy that generates plays which satisfy the infinite constraint, (*ii*) the existential problem, where the problem consists in synthesizing a schedule satisfying the constraint, and (*iii*) the universal problem, where the problem is to decide whether all infinite runs satisfy the constraint. Also, we consider three problems that we denote L,[8] when the upper bound is infinite (denoted $U = \infty$ in the example), $L + U$ when there is also an upper bound, and $L + W$ when there is also a weak upper bound. The various results that we have obtained so far are summarized in the following table, and are joint works with Uli Fahrenberg (Aalborg Universitet, Denmark), Kim G. Larsen,

[8]This is because we still have the lower bound 0 as a constraint.

Nicolas Markey, and Jiří Srba (Aalborg Universitet, Denmark). There are several open problems on which we are currently working.

	game problem		existential problem		universal problem	
	0 clock	1 clock	0 clock	1 clock	0 clock	1 clock
L	$\in$ UP $\cap$ coUP PTIME-hard		$\in$ PTIME	$\in$ PTIME	$\in$ PTIME	$\in$ PTIME
$L+W$	$\in$ NP $\cap$ coNP PTIME-hard		$\in$ PTIME	$\in$ PTIME	$\in$ PTIME	$\in$ PTIME
$L+U$	EXPTIME-complete	Undecidable	$\in$ PSPACE NP-hard		$\in$ PTIME	

Let us mention that the 0-clock game problem with no upper bound is actually equivalent to the mean-cost game problem, for which the precise complexity is a well-known open problem.

Chapter 6

Probabilities in timed automata

6.1 Introduction

The initial motivation for that work was the problem of the implementability of timed systems, and in particular the adequacy of mathematical models like timed automata to systems with real-time constraints. Then, after the first developments with that motivation in mind, a second motivation has risen, that we could actually develop an interesting model for systems with both timing and probabilistic constraints. We will now explain with some more details the two motivations.

Timed automata: an idealised (mathematical) model. Timed automata are a well-established formalism for the modelling and analysis of timed systems. However, like most models used in model checking, timed automata are an idealised mathematical model. In particular it has infinite precision, instantaneous events and communications, *etc.* In real-life systems, the precision of digital clocks is finite, events and communications are not instantaneous, *etc.* The assumptions made in the mathematical model would somewhat need to be relaxed if one wants the model be fair with the real system.

Recently, some research has been devoted to propose alternative semantics to timed automata that provide more realistic operational models for real-time systems. Let us first mention the Almost ASAP semantics introduced in [DDR04]. This AASAP semantics somewhat relaxes the constraints and precision of clocks. However, it induces a very strong notion of robustness [DDMR04, ALM05, BMR06, BMR08, DDMR08], interesting and suitable for really critical systems (like rockets or X-by-wire systems in cars), but maybe too strong for less critical systems (like mobile phones or net-

work applications). In the same vein, another 'robust semantics', based on a notion of tube acceptance, has been proposed in [GHJ97, HR00]. In this framework, a metric is put on the set of traces of the timed automaton, and roughly, a trace is robustly accepted if a tube around that trace is accepted in the classical way. This language-focused notion of acceptance is not completely satisfactory, because it does not take into account the structure of the automaton.

Varacca and Völzer proposed in [VV06] a probabilistic framework for finite-state (time-abstract) systems to overcome side-effects of modelling. They use probabilities to define the notion of being fairly correct as having probability zero to fail, when every non-deterministic choice has been transformed into a 'reasonable' probabilistic choice. In that framework, behaviours that are unlikely will be ignored, and we will say that properties that are unlikely to be violated are actually (almost-surely) satisfied by the system. Moreover, in their framework, a system is fairly correct with respect to some property if and only if the set of traces satisfying that property in the system is topologically large (for the classical Cantor topology defined over the set of infinite paths), which somehow attests the relevance of this notion of fair correctness.

In this chapter, we address both motivations, ruling out unlikely sequences of transitions (as in the approach of [VV06]) and ruling out unlikely events (from a time point-of-view, as in the implementability paradigm discussed above). In order to do so, we propose a probabilistic semantics to timed automata, that randomises both delays and events, and consider various model-checking problems for that semantics. The first problem is the almost-sure model-checking problem, which asks whether a property is satisfied with probability one in a timed automaton under the probabilistic semantics. In that context, we have already proven that the almost-sure model-checking problem against ω-regular properties is decidable for *single-clock* timed automata. The algorithm that we have proposed in the one-clock framework is however not correct for two-clock timed automata, leaving a wide range of open questions. These developments on the almost-sure model-checking have been published in [BBB$^+$07, BBB$^+$08a], and are joint works with Christel Baier, Nathalie Bertrand, Thomas Brihaye, and Marcus Größer. Then we have gone one step further and studied the quantitative model-checking problem, which asks what is the probability for a property to be satisfied in a timed automaton. Also, computing approximations of that probability, or deciding whether this probability is below or above a fixed threshold, are interesting questions. They have been addressed in [BBBM08] and we have proven the computability of these questions in a restricted subclass of single-clock timed automata. This is joint work with

Nathalie Bertrand, Thomas Brihaye, and Nicolas Markey.

Probabilistic and real-time systems. The previous probabilistic semantics that we have given to timed automata can be viewed as a new model with probabilistic and real-time features. We will see later that it actually extends the classical discrete-time and continuous-time Markov chain models (we now write DTMC, respectively CTMC for short), intensively studied both by mathematicians and by computer scientists, see [Fel68]. This raises another interest for our model (and extensions thereof), because DTMCs and CTMCs are already an interesting model to represent systems including probabilistic aspects [HKMS03]. Furthermore, there are more and more complex systems involving real-time constraints, probabilistic behaviours, non-determinism, *etc.*, for instance networks and communication protocols. Let us mention the IEEE 1394 root contention protocol that has been extensively analysed in the early 21st century, see [Sto03] for a brief overview of the various models, and which has many of the above-mentioned characteristics. In all these studies, the proposed model is a trade-off between several aspects of the protocol. There is thus a hope that our model and several-player extensions thereof will be expressive enough to capture systems like the one mentioned above.

There are quite a bunch of models in the literature that mix probabilistic and real-time aspects. One of the most famous models in the model-checking community is the probabilistic timed automata model [KNSS02], for which the tool PRISM is developed [KNP04]. In this model, the time is controlled as in a classical timed automaton, and discrete distributions are put over edges. This model is different from the one we propose, as we randomise delays whereas they do not. However they are able to verify on that model a large class of properties, including those expressible in PTCTL, the probabilistic and timed extension of CTL.

The model we are aware of that is the closest to ours is the model of probabilistic real-time systems that has been proposed in [ACD91, ACD92]. In this model, a clock is associated with an event, and when an event is scheduled, its clock is randomly reinitialised within a fixed interval. This event terminates when the clock (which is 'count-down', *i.e.* decreases) reaches zero. Our model differs from that model in (*i*) the way timing constraints are expressed (we allow the general form of timed automata), and (*ii*) the way randomisation of delays is made (in [ACD91, ACD92], variables are randomly reinitialised, whereas in our work, each time a delay has to be chosen, it is randomly chosen). This yields pretty different properties and results: for instance, in their model, clocks behave rather independently (see [ACD91,

page 8]), and this is not the case in our model (see page 114 where a convergence phenomenon is pointed out, that can *a priori* not be reproduced in the [ACD91] framework).

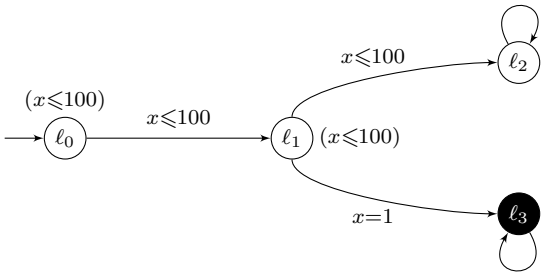
The model presented in [KNSS00] generalises both the models of [KNSS02] and of [ACD91]. As in [ACD91], the randomisation over delays is made when reinitialising clocks, but a general structure of a timed automaton is allowed to model constraints over time. An approximation algorithm based on a refinement of the region graph is proposed to model-check PTCTL. This is not quite related to our work, because the properties that are considered are also different.

Finally, there are tremendously many works on systems that combine timing constraints and (continuous and discrete) probabilities, in which the motivation is less algorithmic than in the previously mentioned works, but is more concerned with behavioural equivalences. Those include interactive generalised semi-Markov processes [BG02], labelled Markov processes [DP03], stochastic automata [DK05], *etc.* The motivation of those works are not the same as ours, even though some of these models have been used as a very general description language MoDeST [BDHK06] for verification purposes [BHK07]. However, it will clearly be interesting to understand better the similarities and the differences between these models and ours.

6.2 Adding probabilities to timed automata

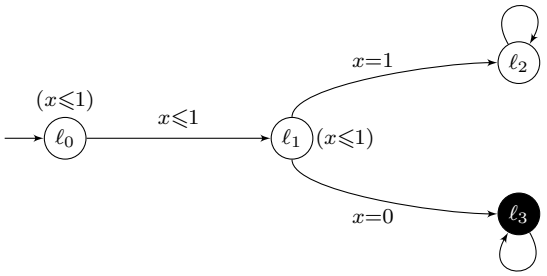
6.2.1 Some intuition

In the timed systems verification process, it may be the case that a property is violated due to really unlikely events. Consider for instance the following automaton:



The property expressed as the LTL formula '$\mathbf{G}\ \neg$black' is not satisfied because there is a way (there are actually several ways) to violate it in that automaton, namely by taking the transition between ℓ_0 and ℓ_1 when $x = .5$ and the transition from ℓ_1 to ℓ_3 when $x = 1$. However, the counter-examples to that

property are really unlikely, and it does not seem unfair to say that the above automaton 'almost-surely' satisfies the property '$\mathbf{G}\,\neg$black' (with no formal meaning for almost-surely for now). From that example, one could say that we can only remove transitions with equality constraints, but we now give another example:



In this automaton, from ℓ_1, there are only guards with equality constraints, so should we remove those two transitions? Of course not, because otherwise the behaviour of the system would really be changed (in particular it would now be blocking whereas it was non-blocking before). So maybe we can then keep the transitions with equality constraints in case there are no other transitions? Why not... but this is maybe not very satisfactory: indeed, in the above automaton, the only way to be able to take the transition from ℓ_1 to ℓ_3 (hence to violate the property '$\mathbf{G}\,\neg$black') is to take the transition from ℓ_0 to ℓ_1 when $x = 0$. This is very unlikely, and intuitively we would also like to say that the above automaton 'almost-surely' satisfies the property '$\mathbf{G}\,\neg$black'.

Our aim is to formalise these intuitions, and to use probabilities to remove unlikely (sequences of) events.

For the rest of the section, we assume $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L})$ is a non-blocking timed automaton (this non-blocking hypothesis will ensure in particular the probability measure be well-defined). We will define an alternative semantics to that timed automaton, that will provide a measure on the set of runs of $\mathcal{A}$. This alternative semantics aims at measuring the likelihood of events and of sets of runs: we will thus assign continuous probability distributions to delays, and discrete probability distributions to transitions that can be taken.

6.2.2 Preliminaries

We first define the objects that we will measure, *i.e.*, those objects that will belong to the σ-algebra on which our probability measure will be defined.

Let $s \in L \times \mathbb{R}^X_{\geqslant 0}$ be a configuration of $\mathcal{A}$ and $(e_i)_{1 \leqslant i \leqslant n}$ be a finite sequence of edges. If $\mathcal{C}$ is a constraint over the n variables $(\tau_i)_{1 \leqslant i \leqslant n}$, the *(symbolic) path* starting from s, determined by $(e_i)_{1 \leqslant i \leqslant n}$, and constrained by $\mathcal{C}$, is the following set of finite runs:

$$\pi_{\mathcal{C}}(s, e_1 \dots e_n) = \{s \xrightarrow{t_1, e_1} s_1 \dots \xrightarrow{t_n, e_n} s_n \in \mathsf{Runs_f}(\mathcal{A}, s) \mid (t_i)_{1 \leqslant i \leqslant n} \models \mathcal{C}\}$$

This is the set of runs that can be read from configuration s on the sequence of edges $(e_i)_{1 \leqslant i \leqslant p}$ so that the delays between events moreover satisfy the constraint $\mathcal{C}$. If $\mathcal{C}$ is equivalent to 'true', we simply write $\pi(s, e_1 \dots e_n)$. Let $\pi = \pi_{\mathcal{C}}(s, e_1 \dots e_n)$ be a finite symbolic path, we define the *cylinder* generated by π as:

$$\mathsf{Cyl}(\pi) = \{\varrho \in \mathsf{Runs}(\mathcal{A}, s) \mid \exists \varrho' \in \pi \text{ such that } \varrho' \text{ is a finite prefix of } \varrho\}$$

It is the set of infinite runs that have a finite prefix in π. Given s a configuration of $\mathcal{A}$ and e an edge, we define $I(s, e) = \{\tau \in \mathbb{R}_{\geqslant 0} \mid \exists s' \in L \times \mathbb{R}^X_{\geqslant 0} \text{ such that } s \xrightarrow{\tau, e} s'\}$ the set of delays that enable edge e from s, and we define $I(s) = \bigcup_{e \in E} I(s, e)$ the set of delays from which an edge can be fired.

6.2.3 The probabilistic semantics

We assume probability distributions are given from every valid configuration s of $\mathcal{A}$ both over delays and over enabled edges. Formally, for every configuration s of $\mathcal{A}$, we write μ_s for the probability measure over possible delays from s, *i.e.*, over $I(s)$ ($\mathbb{R}_{\geqslant 0}$ is equipped with the standard Borel σ-algebra). It must satisfy several natural requirements:

- $\mu_s(I(s)) = \mu_s(\mathbb{R}_{\geqslant 0}) = 1$;[1]
- Denoting λ the Lebesgue measure, if $\lambda(I(s)) > 0$, μ_s is equivalent[2] to λ on $I(s)$; Otherwise, μ_s is equivalent on $I(s)$ to the uniform distribution over points of $I(s)$. This condition denotes some kind of fairness with respect to enabled transitions, in that we cannot disallow one transition by putting a probability 0 to delays enabling that transition;
- There are some other technical requirements that we do not expose here (see [BBB+08a]).

[1] Note that this is possible, as we assume $\mathcal{A}$ is non-blocking, hence $I(s) \neq \emptyset$ for every configuration s of $\mathcal{A}$.

[2] Two measures μ and μ' are *equivalent* whenever for each measurable set A, $\mu(A) = 0 \Leftrightarrow \mu'(A) = 0$.

***Remark* 6.1** All the above requirements can be easily satisfied. For instance, a timed automaton equipped with uniform (respectively exponential) distributions on bounded (respectively unbounded) intervals satisfy these conditions. If we assume exponential distributions on unbounded intervals, it is moreover required that the transition rate be bounded, like in [DP03], to avoid strange phenomena. We recall some basic and useful vocabulary in probability theory in table 6.1. ⌟

A measure μ over $\mathbb{R}$ has *density* f (with respect to the Lebesgue measure) if for every Borel-measurable set A, $\mu(A) = \int_{t \in A} f(t)\,\mathrm{d}t$ (this requires f to be a measurable function).

A probability measure μ is said *uniform over the interval* $[a, b]$ (with $a < b$) whenever it has density function $t \mapsto \dfrac{1}{b-a}$ over $[a, b]$ and $t \mapsto 0$ over $\mathbb{R} \setminus [a, b]$. Hence, if λ denotes the classical Lebesgue measure and if I is an interval, then $\mu(I) = \dfrac{1}{b-a} \cdot \lambda(I \cap [a, b])$.

A probability measure μ is an *exponential distribution* over the interval $[\alpha, +\infty)$ if there exists a rational $\beta > 0$ such that μ has density $t \mapsto \beta \cdot e^{-\beta(t-\alpha)}$ over $[\alpha, +\infty)$ and $t \mapsto 0$ over $(-\infty, \alpha)$. The value β is the *rate* of the distribution.

Table 6.1: Probabilistic vocabulary

For every valid configuration s of $\mathcal{A}$, we also assume a probability distribution p_s over edges, such that for every edge e, $p_s(e) > 0$ if and only if e is enabled in s. Moreover, to simplify, we assume that p_s is given by weights on transitions, as it is classically done for resolving non-determinism: we associate with each edge e a rational weight $w(e) > 0$, and for every state s, for every edge e, $p_s(e) = 0$ if e is not enabled in s, and $p_s(e) = w(e)/(\sum_{e' \text{ enabled in } s} w(e'))$ otherwise. As a consequence, if s and s' are region equivalent, then for every edge e, $p_s(e) = p_{s'}(e)$. We then define a measure over finite (unconstrained) symbolic paths from configuration s as $\mathbb{P}_{\mathcal{A}}(\pi(s)) = 1$ for the trivial symbolic path from s, and if $\pi(s, e_1 \dots e_n)$ is a finite symbolic path, we let

$$\mathbb{P}_{\mathcal{A}}(\pi(s, e_1 \dots e_n)) = \int_{t \in I(s,e_1)} p_{s+t}(e_1) \cdot \mathbb{P}_{\mathcal{A}}(\pi(s_t^{e_1}, e_2 \dots e_n))\,\mathrm{d}\mu_s(t)$$

where $s \xrightarrow{t} (s+t) \xrightarrow{e_1} s_t^{e_1}$. The intuition behind the formula for $\mathbb{P}_{\mathcal{A}}$ is the following: the probability of taking transition e_1 after t time units coincides with the probability of waiting t time units and then choosing e_1 among

the enabled transitions, *i.e.*, '$p_{s+t}(e_1)\,\mathrm{d}\mu_s(t)$'. Note that, time passage and actions are independent events.

The value $\mathbb{P}_{\mathcal{A}}(\pi(s, e_1 \dots e_n))$ is the result of n successive one-dimensional integrals, but it can also be viewed as the result of an n-dimensional integral. Hence, we can easily extend the above definition to finite constrained paths $\pi_{\mathcal{C}}(s, e_1 \dots e_n)$ when $\mathcal{C}$ is Borel-measurable. This extension to constrained paths will allow to express (and thus measure) various and rather complex sets of paths, for instance Zeno runs (see example 6.2 below). The measure $\mathbb{P}_{\mathcal{A}}$ can then be defined on cylinders, letting $\mathbb{P}_{\mathcal{A}}(\mathsf{Cyl}(\pi)) = \mathbb{P}_{\mathcal{A}}(\pi)$ if π is a finite (constrained) symbolic path. Finally we extend $\mathbb{P}_{\mathcal{A}}$ in a standard and unique way to the σ-algebra $\Omega^s_{\mathcal{A}}$ generated by these cylinders.

***Example* 6.2** The set of Zeno runs $\mathsf{Zeno}(s)$ from configuration s in $\mathcal{A}$ is in the σ-algebra $\Omega^s_{\mathcal{A}}$ because it can be written as:

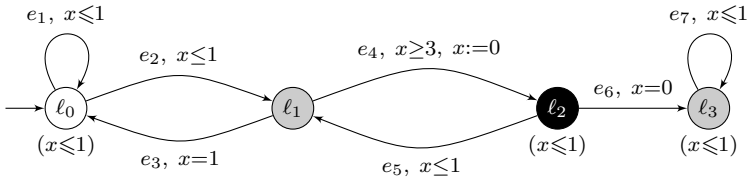
$$\mathsf{Zeno}(s) = \bigcup_{M\in\mathbb{N}} \bigcap_{n\in\mathbb{N}} \bigcup_{(e_1,\dots,e_n)\in E^n} \pi_{\mathcal{C}_{M,n}}(s, e_1 \dots e_n)$$

where $\mathcal{C}_{M,n}$ is the constraint $\sum_{i=1}^{n} \tau_i \leqslant M$. Indeed, an infinite run $\varrho = s_0 \xrightarrow{t_1,e_1} s_1 \cdots \xrightarrow{t_i,e_i} s_i \cdots$ is in $\mathsf{Zeno}(s)$ if there exists $M \in \mathbb{N}$ such that $\sum_{i\in\mathbb{N}} t_i \leqslant M$. ⌟

This non-standard semantics for timed automata enjoys the following basic property, that justifies its name of *probabilistic semantics*, but whose proof requires a careful analysis, see [BBB+08b].

☞ **Proposition 6.3 ([BBB+08a])** *Let $\mathcal{A}$ be a timed automaton. For every configuration s of $\mathcal{A}$, $\mathbb{P}_{\mathcal{A}}$ is a probability measure over $(\mathsf{Runs}(\mathcal{A}, s), \Omega^s_{\mathcal{A}})$.*

***Example* 6.4** We consider the timed automaton $\mathcal{A}$ depicted on the figure below (we assume atomic propositions are 'white', 'grey', and 'black'). We furthermore assume uniform distributions over edges (each edge has weight 1), uniform distributions over possible delays from all valid configurations with location ℓ_0, ℓ_2 or ℓ_3, and exponential distribution with density $t \mapsto e^{-t}$ over delays from all valid configurations with location ℓ_1.



If $s_0 = (\ell_0, 0)$ is the initial state, then

$$\begin{cases} \mathbb{P}_{\mathcal{A}}(\mathsf{Cyl}(\pi(s_0, e_1 e_1))) & = & \frac{1}{4} \\ \mathbb{P}_{\mathcal{A}}(\mathsf{Cyl}(\pi_{\tau_2<5}(s_0, e_2 e_4))) & = & \frac{1}{2}(1 + e^{-3} - e^{-2}) \end{cases}$$

Details of the computations are given in table 6.2. ⌟

$$\begin{aligned} \mathbb{P}_{\mathcal{A}}(\mathsf{Cyl}(\pi_{\tau_2<5}(s_0, e_2 e_4))) &= \int_{t_1=0}^{1} \frac{1}{2} \int_{t_2=3-t_1}^{5} e^{-(t_2-3+t_1)} \, \mathrm{d}t_2 \, \mathrm{d}t_1 \\ &= \int_{t_1=0}^{1} \frac{1}{2} \left[-e^{-(t_2-3+t_1)} \right]_{t_2=3-t_1}^{5} \, \mathrm{d}t_1 \\ &= \int_{t_1=0}^{1} \frac{1}{2} \left(1 - e^{-(2+t_1)} \right) \, \mathrm{d}t_1 \\ &= \left[\frac{1}{2} \left(t_1 + e^{-(2+t_1)} \right) \right]_{t_1=0}^{1} \\ &= \frac{1}{2} \left(1 + e^{-3} - e^{-2} \right) \\ &\approx 0.46 \end{aligned}$$

From $s_0 = (\ell_0, 0)$, the distribution over delays is the uniform probability measure over interval $[0, 1]$. The $\frac{1}{2}$ term is due to the fact that each edge e_1 and e_2 are equally probable. Then, from a state (ℓ_1, t_1) the distribution is the exponential distribution on interval $[3 - t_1, +\infty)$ (because the constraint $x \geqslant 3$ will be enabled $3 - t_1$ time units later), hence has density $t \mapsto e^{-(t-3+t_1)}$. We add the constraint that $t_2 < 5$, which yields the first line of the equations above.

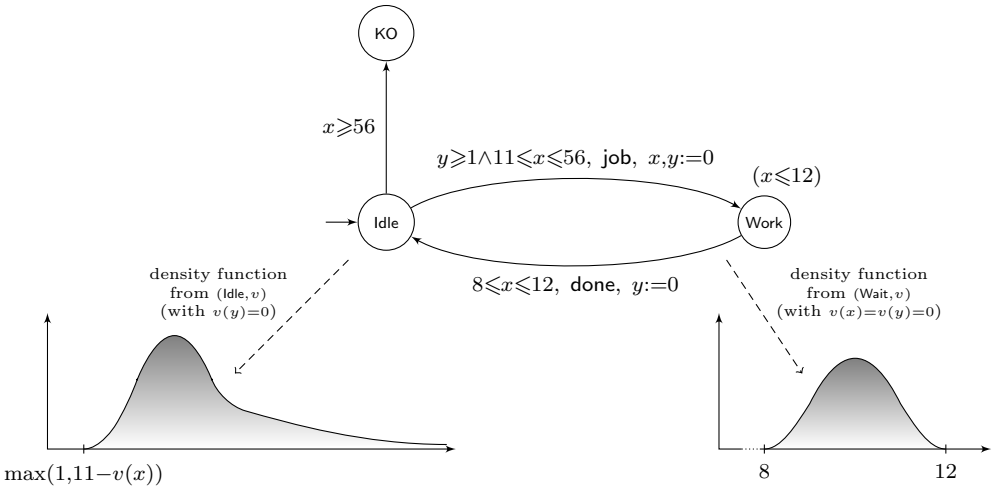
Table 6.2: Details of the computation for example 6.4

***Remark* 6.5** We have mentioned in the introduction of the chapter that our model generalises CTMCs, and we briefly explain why now. A CTMC is nothing else than a single-clock timed automaton in which (*i*) on all transitions, the clock constraint is trivial, and the clock is reset, and (*ii*) for every location ℓ, there is a rate $\beta_\ell > 0$ such that from every configuration with location ℓ, the probability measure over delays is an exponential distribution with rate β_ℓ. ⌟

Given an infinite symbolic path π in $\mathcal{A}$, and an ω-regular property φ, either all concretizations of π (*i.e.*, runs $\varrho \in \pi$) satisfy φ, or they all do not satisfy φ. Hence, the set $\{\varrho \in \mathsf{Runs}(\mathcal{A}, s_0) \mid \varrho \models \varphi\}$ is measurable (in $\Omega_{\mathcal{A}}^{s_0}$) [Var85]. In the sequel, we write $\mathbb{P}_{\mathcal{A}}(s_0 \models \varphi)$ or even simply $\mathbb{P}_{\mathcal{A}}(\varphi)$ if s_0 is the initial configuration of $\mathcal{A}$ for $\mathbb{P}_{\mathcal{A}}\{\varrho \in \mathsf{Runs}(\mathcal{A}, s_0) \mid \varrho \models \varphi\}$.

6.2.4 A small example

We assume a (simple) printer receiving pages to be printed. The delay between two pages that are processed is at least 11 time units, and at most 56 time units (after 56 time units, the printer goes to the out-of-order mode). The printer needs furthermore to be idle for at least 1 time unit between it finishes printing a page and can process the next page. Also the time required for printing one page lies between 8 and 12 time units. When idle, the arrival time of the next page is given by the density function given on the left. The time taken to print a page is given by the density function depicted on the right. Questions that could be asked on this system are for instance: given a 10-page long paper, (*i*) what is the probability that it will be printed in no more than 96 time units? and (*ii*) is the probability the printer becomes out-of-order during the printing less than 10%?



This example is of course a toy example, but we believe we can model interesting phenomena using this semantics, for instance message losses. Of course, density functions for the distributions are then (a difficult) part of the modelling work.

In the two next sections, we will be interested in several model-checking problems under the probabilistic semantics. In section 6.3, we will consider the qualitative model-checking problem, and in section 6.5, we will extend our study to some quantitative model-checking problems.

6.3 The qualitative model-checking problem

In this section, we will be interested in qualitative questions about the probabilistic semantics.

6.3.1 Definition

Definition 6.6 *Let φ be an ω-regular property and $\mathcal{A}$ a timed automaton. We say that $\mathcal{A}$* almost-surely satisfies *φ, and we then write $\mathcal{A} \mid\!\approx_{\mathbb{P}} \varphi$, whenever $\mathbb{P}_{\mathcal{A}}(s_0 \models \varphi) = 1$, where s_0 is the initial configuration of $\mathcal{A}$. The* almost-sure model-checking problem *asks, given $\mathcal{A}$ and φ, whether $\mathcal{A} \mid\!\approx_{\mathbb{P}} \varphi$.*

***Remark* 6.7** Note that all other classical qualitative questions (is the probability of φ in $\mathcal{A}$ equal to 0? Or positive? Or strictly smaller than 1?) reduce to the almost-sure question that we have defined. Indeed,

$$\begin{cases} \mathbb{P}_{\mathcal{A}}(s_0 \models \varphi) = 0 & \Leftrightarrow \quad \mathbb{P}_{\mathcal{A}}(s_0 \models \neg\varphi) = 1 \\ \mathbb{P}_{\mathcal{A}}(s_0 \models \varphi) > 0 & \Leftrightarrow \quad \neg\Big(\mathbb{P}_{\mathcal{A}}(s_0 \models \neg\varphi) = 1\Big) \\ \mathbb{P}_{\mathcal{A}}(s_0 \models \varphi) < 1 & \Leftrightarrow \quad \neg\Big(\mathbb{P}_{\mathcal{A}}(s_0 \models \varphi) = 1\Big) \end{cases}$$

and if φ is ω-regular, then so is $\neg\varphi$. ⌟

***Example* 6.8** As a first example, we consider the two examples presented at the beginning of section 6.2. In the two timed automata, we put uniform distributions over delays from all valid configurations with location ℓ_0 or ℓ_1. Then it is not difficult to check that in both cases, $\mathbb{P}(s_0 \models \mathbf{G}\,\neg\text{black}) = 0$. ⌟

***Example* 6.9** Consider again the timed automaton $\mathcal{A}$ described in example 6.4 with the same distributions over delays and edges. Let φ be the ω-regular property defined by the LTL formula '$\mathbf{F}\,(\text{grey} \wedge \mathbf{G}\,(\text{grey} \Rightarrow \mathbf{F}\,\text{black}))$'. Then, $\mathcal{A} \mid\!\approx_{\mathbb{P}} \varphi$. Indeed, from configuration (ℓ_0, ν) with $0 \leq \nu \leq 1$, the probability of firing e_2 (after some delay) is always 0.5 (guards of e_1 and e_2 are the same, there is thus a uniform distribution over both edges), thus the location ℓ_1 is reached with probability 1. In ℓ_1, the transition e_3 will unlikely happen, because its guard $x = 1$ is much too "small" compared to the guard $x \geqslant 3$ of the transition e_4. The same phenomenon arises in location ℓ_2 between the transitions e_5 and e_6. In conclusion, the runs of the timed automaton $\mathcal{A}$ (from s_0) are almost surely following sequences of transitions of the form ${e_1}^* e_2 (e_4 e_5)^\omega$. Hence, with probability 1, the formula φ is satisfied. Note that the previous formula is not satisfied with the classical LTL semantics. Indeed several counter-examples to the satisfaction of the formula can be found: '*staying in ℓ_0 forever*', '*reaching ℓ_3*', *etc.* All these counter-examples are unlikely and disappear thanks to the probabilistic interpretation. ⌟

Although the values $\mathbb{P}_{\mathcal{A}}(s_0 \models \varphi)$ depend on the chosen weights $p_s(e)$ and measures μ_s, we will see that for single-clock timed automata the almost-sure satisfaction relation is not affected by a (reasonable) choice of the weights and distributions. This will be crucial for the decidability of the almost-sure model-checking problem. The way we establish the decidability is to build a finite Markov chain that will satisfy almost-surely an ω-regular property if and only if the original timed automaton almost-surely satisfies the initial property.

6.3.2 The algorithm

The algorithm to decide the qualitative model-checking of timed automata is described as Algorithm 1. All steps of the algorithm should be clear, except

Algorithm 1: Algorithm for the qualitative model-checking

Data: A timed automaton $\mathcal{A}$, an ω-regular property φ
Result: Does $\mathcal{A} \not\approx_{\mathbb{P}} \varphi$?

```
1  begin
2  |  Build the region automaton Γ(A) of A;
3  |  Remove unlikely transitions in Γ(A) and non-reachable states;
4  |  Write MC(A) for the resulting structure, interpreted as a finite
   |  Markov chain (with uniform weights on edges);
5  |  if P_{MC(A)}(φ) = 1 then
6  |  |  answer 'Yes';
7  |  else
8  |  |  answer 'No';
9  |  end
10 end
```

line 3. A transition leaving state q of $\Gamma(\mathcal{A})$ is said *unlikely* whenever it is implicitly labelled by a punctual constraint (of the form $x = c$), whereas transitions implicitly labelled with non-punctual constraints (like $x \in (d, d+1)$) can be taken from q as well. Note that this removal can be done syntactically on $\Gamma(\mathcal{A})$. Intuitively, this is because the continuous distribution over delays will give probability 0 to that 'unlikely' transition. Transitions from state q of $\Gamma(\mathcal{A})$ that are constrained by a punctual constraint but so that only punctual constraints label transitions leaving q are not removed because, as all transitions from q, they are likely to be chosen from q.

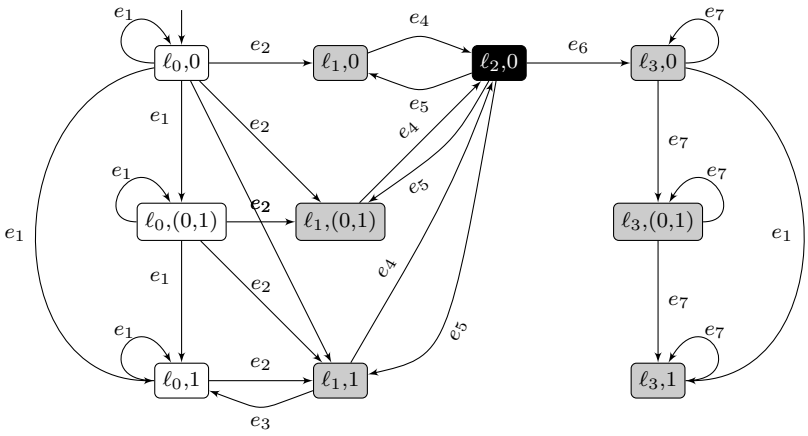
***Example* 6.10** We consider again the automaton $\mathcal{A}$ of example 6.4, and the ω-

regular property φ expressed as the LTL formula '$\mathbf{F}$ (grey $\wedge$ $\mathbf{G}$ (grey $\Rightarrow$ $\mathbf{F}$ black))'. As already argued in example 6.9, we have that

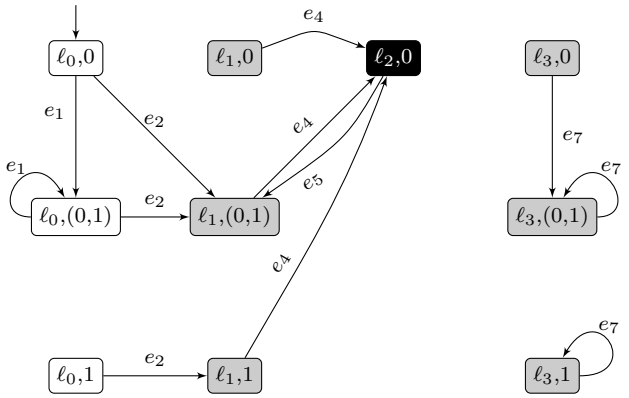
$$\mathcal{A} \not\models \varphi \qquad \text{but} \qquad \mathcal{A} \not\approx_{\mathbb{P}} \varphi$$

because almost-surely, paths are of the form $e_1^* e_2 \big(e_4 e_5\big)^\omega$.

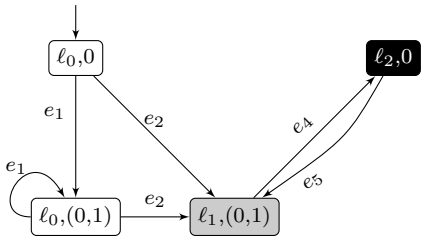
We illustrate Algorithm 1 on this example. First the region automaton $\Gamma(\mathcal{A})$ is constructed:



Then, unlikely transitions are (locally) removed (for instance, transition e_2 that was between $(\ell_0, 0)$ and $(\ell_1, 0)$ in $\Gamma(\mathcal{A})$ has been removed because it was implicitly guarded by the punctual constraint $x = 0$, whereas a larger guard $x \in (0, 1)$ implicitly constrains the transition between $(\ell_0, 0)$ and $(\ell_1, (0, 1))$):



Non-reachable states are removed, and we get the following graph, that we will now interpret as a finite Markov chain $\mathsf{MC}(\mathcal{A})$ (with weight 1 on each edge):



It is easy to be convinced that almost-surely paths in $\mathsf{MC}(\mathcal{A})$ are of the form $e_1^* e_2 (e_4 e_5)^\omega$, and that property φ almost-surely holds in $\mathsf{MC}(\mathcal{A})$. Thus, Algorithm 1 answers that $\mathcal{A}$ almost-surely satisfies property φ. ⌟

The correctness of Algorithm 1 relies on the following proposition.

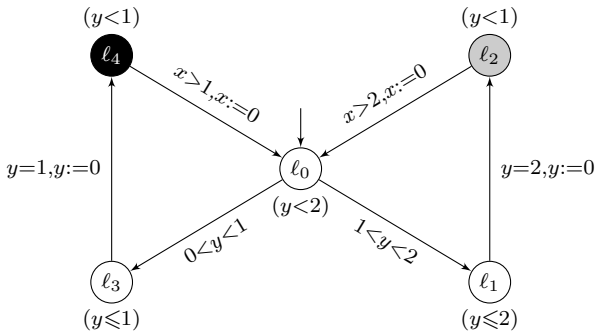
☞ **Proposition 6.11 ([BBB$^+$08a])** *Let $\mathcal{A}$ be a **single-clock** timed automaton, and φ an ω-regular property. Then,*

$$\mathcal{A} \mathrel{\not\approx} \varphi \qquad \Leftrightarrow \qquad \mathbb{P}_{\mathsf{MC}(\mathcal{A})}(\varphi) = 1$$

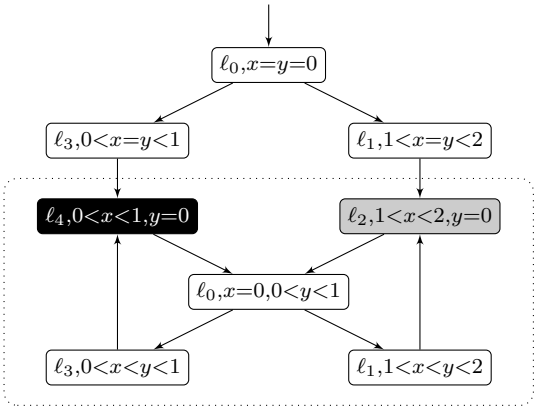
The proof of this proposition is non-trivial, we will not give details here. Let us just mention that it relies on the equivalence with a topological semantics for timed automata based on the notion of *largeness*. The topology that is put on runs of timed automata extends the classical Cantor topology in finite automata, used for instance in [VV06]. The technicalities heavily rely on the topological Banach-Mazur games that characterise large sets using winning strategies [Oxt57].

6.3.3 A two-clock counter-example

Proposition 6.11 does not extend to the class of timed automata with two clocks, and Algorithm 1, though rather intuitive and natural, is actually not correct for the class of two-clock timed automata. Indeed, consider the following timed automaton $\mathcal{A}$ (that we had already considered on page 25) with two clocks and the ω-regular property φ given by the LTL formula '**G F** grey $\wedge$ **G F** black'.



We assume uniform distributions over delays from every valid configuration. Using basic mathematical arguments that we will not detail here (see [BBB+08b]), we can prove that, if $s_0 = (\ell_0, \mathbf{0}_X)$, $\mathbb{P}_{\mathcal{A}}(s_0 \models \varphi) < 1$ (it is even the case that $\mathbb{P}_{\mathcal{A}}(s_0 \models \varphi) = 0$). However, the finite Markov chain $\mathsf{MC}(\mathcal{A})$, as constructed in Algorithm 1 is:



It is not difficult to check that this finite Markov chain almost-surely satisfies φ, because there is a single BSCC[3] (surrounded by the dotted line) that contains a grey state and a black state. Hence, Algorithm 1 is not correct for two-clock timed automata.

6.3.4 Summary

The main result of this section can be stated as follows:

☞ **Theorem 6.12 ([BBB+08a])** *The almost-sure model-checking problem of ω-regular (respectively* LTL*) properties is* NLOGSPACE*-complete (respectively* PSPACE*-complete) for single-clock timed automata.*

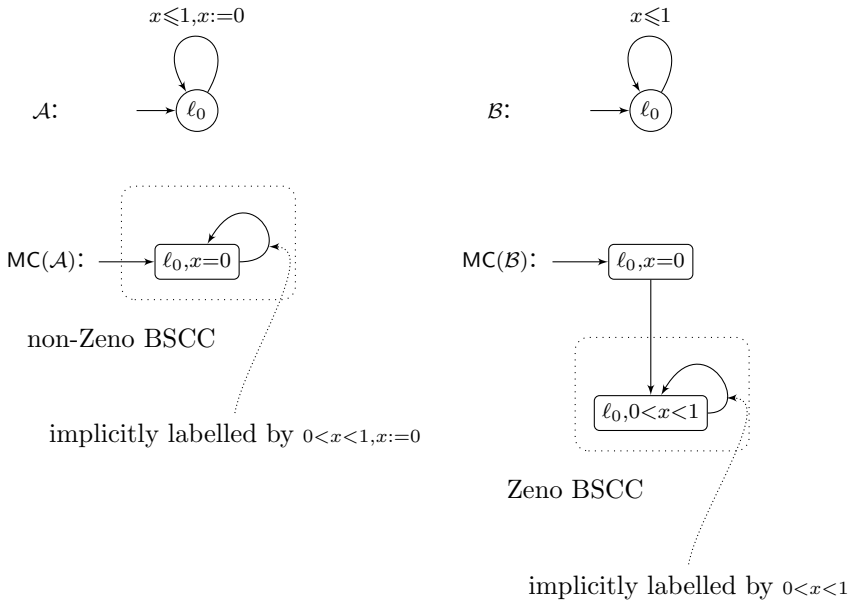
***Remark* 6.13** As a side-result of the correctness of Algorithm 1, we get that the almost-sure satisfaction of ω-regular is independent of the choice of the probability distributions! Of course, one needs to be reasonable, and to satisfy the 'fairness' hypotheses mentioned in section 6.2. ⌟

6.4 An interesting notion of non-Zenoness

Let $\mathcal{A}$ be a timed automaton and s_0 the initial configuration of $\mathcal{A}$. We have already seen that the set of Zeno runs from s_0 are measurable (they belong

[3]BSCC stands for 'bottom strongly connected component', and it is an SCC that cannot be left.

to the σ-algebra $\Omega_{\mathcal{A}}^{s_0}$). However, the Zenoness is not an ω-regular property, hence the previous study does not apply. However, for single-clock timed automata, the finite Markov chain is still of a great help, in that we can prove that the probability of being Zeno is 0 from the initial configuration s_0 if and only if all BSCCs in $\mathsf{MC}(\mathcal{A})$ are either unbounded, or reset the clock at least once (in that case we say that the BSCC is not purely Zeno).[4] For instance, consider the two following timed automata, $\mathcal{A}$ and $\mathcal{B}$. We have that $\mathbb{P}_{\mathcal{A}}(\mathsf{Zeno}) = 0$ whereas $\mathbb{P}_{\mathcal{B}}(\mathsf{Zeno}) = 1$. Indeed, the unique BSCC of $\mathsf{MC}(\mathcal{A})$ is not purely Zeno whereas the unique BSCC of $\mathsf{MC}(\mathcal{B})$ is purely Zeno. Note that this somehow coincides with the intuition that only few runs of $\mathcal{A}$ are Zeno, whereas all runs of $\mathcal{B}$ are Zeno.



☞ **Proposition 6.14 ([BBB+08a])** *Given a single-clock timed automaton $\mathcal{A}$ with initial configuration s_0, we can decide in* NLOGSPACE *whether*

$$\mathbb{P}_{\mathcal{A}}(\mathsf{Zeno}(s_0)) = 0$$

This yields an interesting notion of *non-Zeno* timed automaton $\mathcal{A}$ when $\mathbb{P}_{\mathcal{A}}(\mathsf{Zeno}(s_0)) = 0$. In such an automaton, there may be Zeno runs, but they are unlikely to happen. Note that any non-Zeno hypothesis encountered in the literature, like the strongly non-Zenoness assumption of [AMPS98], is more restrictive than our condition.

[4] A BSCC of $\mathsf{MC}(\mathcal{A})$ resets the clock at least once whenever there is at least a transition that is implicitly labelled by some constraint implying $x > 0$ and that resets the clock x.

6.5 The quantitative model-checking problem

The previous finite Markov chain abstraction is correct for checking qualitative properties in single-clock timed automata, but it does not preserve the precise values of the probabilities. Thus, in general, it cannot be used for answering quantitative questions.

6.5.1 Definitions

In this section, we are interested in the following quantitative model-checking questions. Given a timed automaton $\mathcal{A}$ with initial configuration s_0 and an ω-regular property φ, we want to compute the value $\mathbb{P}_{\mathcal{A}}(s_0 \models \varphi)$ or/and for every $\varepsilon > 0$, ε-approximations thereof. Also, we are interested in the following decision problem, called the *threshold problem*: given a threshold $c \in [0,1] \cap \mathbb{Q}$ and a comparison operator $\sim \in \{<, \leqslant, =, \geqslant, >\}$, decide whether $\mathbb{P}_{\mathcal{A}}(s_0 \models \varphi) \sim c$. All these questions are interesting to measure the quality of a model with respect to a property. For instance in a network application modelled as a timed automaton $\mathcal{A}$, if φ denotes the property that every message which is sent is eventually received, we could ask whether $\mathbb{P}_{\mathcal{A}}(s_0 \models \varphi) > 0.95$, which says that the probability of φ is larger than 95%.

In finite Markov chains (and also in CTMCs), answering all the above questions are easy, as we know that for every finite Markov chain and for every ω-regular property, we can build a system of linear equations (with rational coefficients) so that a solution to that system corresponds to the probability the ω-regular property be satisfied, see [CY95, Bré99] for details. The correctness of the method relies on the Markov property, which implies in particular that, if $\varrho = \varrho_1 \cdot \varrho_2$ is the concatenation of two paths, the probability of ϱ is the product of the probabilities of ϱ_1 and of ϱ_2 (the probability over finite paths is multiplicative). In our framework, this is no more the case, computing the probability of a symbolic path cannot be done by computing 'the probability of each transition' separately, and by multiplying the results. For instance, in example 6.4 on page 108, the probability of the symbolic path $\pi_{\tau_2<5}(s_0, e_2e_4)$ has to be computed as two nested integrals, and its computation cannot be decomposed into the computation of two independent integrals, one corresponding to edge e_2 and the other corresponding to edge e_4. A way to become 'multiplicative' is to restrict to a single clock and to reset the unique clock, because in that case, the computation of multiple integrals becomes independent. We will use this idea to develop a method for solving quantitative model-checking questions.

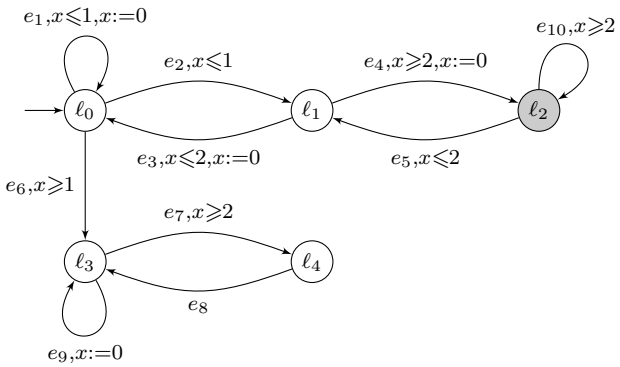
6.5.2 Computability results

Following the hypotheses made for the simpler qualitative model-checking problem, we restrict to the single-clock framework. Let $\mathcal{A}$ be a single-clock timed automaton. We furthermore assume that the following conditions are met: for every reachable and valid configuration s of $\mathcal{A}$, $I(s) = \mathbb{R}_{\geqslant 0}$; for every location ℓ of $\mathcal{A}$, there exists $\lambda_\ell \in \mathbb{Q}_{>0}$ such that for every configuration s with location ℓ, the probability over delays from s is an exponential distribution with rate λ_ℓ; every cycle in $\Gamma(\mathcal{A})$ either resets the clock, or is unbounded (*i.e.*, all regions encountered along the cycle are unbounded). We denote these additional assumptions by (†).

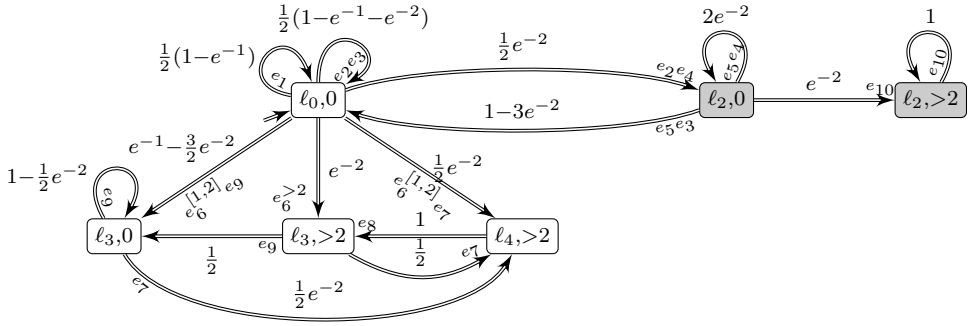
☞ **Theorem 6.15 ([BBBM08])** *For an ω-regular property φ and a single-clock timed automaton $\mathcal{A}$ satisfying the conditions* (†)*, one can (i) compute a closed-form expression for $\mathbb{P}_{\mathcal{A}}(\varphi)$, (ii) compute ε-approximations of $\mathbb{P}_{\mathcal{A}}(\varphi)$ for every $\varepsilon > 0$, and (iii) decide the threshold problem.*

The idea is the following: in parts of the automaton where the clock is unbounded, the automaton behaves like a CTMC, hence the only relevant probabilistic information is given by the weights of the transitions (because the properties we are considering are untimed, hence the distributions over delays do not play any role here); in parts of the automaton where the clock is bounded, we can decompose each run of the automaton into "macro-steps" where one step corresponds to a sub-run starting with the clock reset to 0, and finishing with a resetting edge (these macro-steps have bounded length, due to the assumption that there is no bounded cycle without resets). Given $\mathcal{A}$ a single-clock timed automaton satisfying hypothesis (†), we can thus construct a **finite Markov chain** $\mathsf{MC}'(\mathcal{A})$ that will basically preserve the quantitative properties of the initial system. Values labelling the edges of $\mathsf{MC}'(\mathcal{A})$ correspond to the probability of the corresponding symbolic path in $\mathcal{A}$, and may thus be non-rational, but techniques developed for classical Markov chains can however still be used. We do not enter into the details here but better develop an example. Note that the abstraction $\mathsf{MC}'(\mathcal{A})$ is correct for a larger class of timed automata than those satisfying (†), but that is simpler to focus on that class.

***Example* 6.16** We consider the timed automaton $\mathcal{A}$ depicted on the next page, in which we assume that the probability over delays from every configuration is an exponential distribution with rate 1, and that the weight of each transition is 1.



We abstract this timed automaton into the finite Markov chain $\mathsf{MC}'(\mathcal{A})$ that is depicted below. We explain some of the edges of $\mathsf{MC}'(\mathcal{A})$. The self-loop on $(\ell_0, 0)$ labelled by e_2e_3 is for all the paths read from $(\ell_0, 0)$ over the sequence e_2e_3 (e_3 is a resetting edge). The edge labelled $e_6^{>2}$ between states $(\ell_0, 0)$ and $(\ell_3, > 2)$ are for those paths starting from the configuration $(\ell_0, 0)$ and taking edge e_6, with the additional constraint that we arrive in ℓ_3 with the value of the clock being larger than 2. The edge between $(\ell_3, > 2)$ and $(\ell_4, > 2)$ labelled by e_7 represents all the paths starting from some (ℓ_3, x) with $x > 2$ to state ℓ_4 firing edge e_7 (this part behaves like a CTMC). The values labelling the edges give the probabilities of paths they represent. We explain the computation of the value labelling the edge from $(\ell_0, 0)$ to $(\ell_3, 0)$ in table 6.3.



This finite Markov chain preserves (most of) the quantitative properties of the original timed automaton. For instance, we have that:

$$\begin{aligned}\mathbb{P}_{\mathcal{A}}(s_0 \models \mathbf{F}\ \text{grey}) &= \mathbb{P}_{\mathsf{MC}'(\mathcal{A})}((\ell_0, 0) \models \mathbf{F}\ \text{grey}) \\ &= \frac{1}{2e+1} \\ &\approx 0.16\end{aligned}$$

Due to the hypotheses made on the timed automaton $\mathcal{A}$, labels of the finite Markov chains $\mathsf{MC}'(\mathcal{A})$ are polynomials in e^{-1}, and for any ω-regular

The edge from $(\ell_0, 0)$ to $(\ell_3, 0)$ in $\mathsf{MC}'(\mathcal{A})$, which is labelled by $e_6^{[1,2]}$, corresponds to runs in $\mathcal{A}$ that start from configuration $(\ell_0, 0)$, take the transition e_6 so that the value of clock x when arriving in ℓ_3 is within the interval $[1, 2]$, and then take transition e_9 once (this last move resets the clock). We label this edge by the probability of these runs, which can be computed as follows:

$$\mathbb{P}_{\mathcal{A}}(\pi_{\tau_1 \leqslant 2}(s_0, e_6 e_9)) = \int_{t_1=1}^{2} e^{-t_1} \cdot \left(\int_{t_2=0}^{2-t_1} e^{-t_1} \, \mathrm{d}t_2 + \frac{1}{2} \int_{t_2=2-t_1}^{+\infty} e^{-t_2} \, \mathrm{d}t_2 \right) \mathrm{d}t_1$$

The first integral corresponds to the firing of the edge e_6, which has to be done within the interval $[1, 2]$ from the beginning. There are two cases for firing the edge e_9: either it is taken before the values of x reaches 2 (this corresponds to the first sub-integral, where t_2 is in the interval $[0, 2 - t_1]$), or it is taken when the value of x is larger than or equal to 2 (this corresponds to the second sub-integral, where the factor $\frac{1}{2}$ is due to the fact that edges e_7 and e_9 are equally probable).

Table 6.3: Details of the computation

property φ, the value $\mathbb{P}_{\mathcal{A}}(\varphi)$ is of the form $f(e^{-1})$, where f is a rational function with rational coefficients (we extend classical techniques for finite Markov chains [CY95, Bré99]). Using the Maclaurin development[5] of the exponential function, one can thus get arbitrary approximations of the value of the probability. Furthermore deciding the threshold problem becomes then easy: if $f = \frac{P}{Q}$ (with P and Q polynomials), and if $c \in \mathbb{Q}_{\geqslant 0}$,

$$\begin{aligned} f(e^{-1}) = c &\Leftrightarrow P(e^{-1}) = c \cdot Q(e^{-1}) \\ &\Leftrightarrow P = c \cdot Q \\ &\quad \text{(because } e^{-1} \text{ is a transcendental number)} \end{aligned}$$

Then, deciding $f(e^{-1}) < c$ (or $f(e^{-1}) > c$) can be done using precise enough approximations of the value $f(e^{-1})$.

In this section, we have described the development we have made in the context of quantitative model-checking for our probabilistic semantics. We have obtained some approximation and decidability results, that are a first step towards the computation of more complex quantitative measures in timed automata.

[5] Or the Taylor development, if we stick to the French terminology.

6.6 Going further – current developments

We are currently investigating several research directions on that model, that we briefly discuss.

Expressiveness, comparison with other formalisms As we have mentioned in the introduction, multiple formalisms for real-time and probabilistic systems have been proposed in the literature. It is not clear to us yet how they really compare to our model. We know that our model is more general than continuous-time Markov chains, and that the way we randomize delays is closer to that of interactive generalised semi-Markov processes [BG02], where clocks are running forward, contrary to formalisms like probabilistic timed systems [ACD91, ACD92], probabilistic timed automata [KNSS00], or stochastic automata [DK05] where clocks are 'count-down' (*i.e.* running backward). We think that makes a difference (properties of models seem different), but we are not quite sure yet if one can be expressed in the other one.

Also, we need to investigate further if classical real systems could be modelled in a more accurate way in our framework, and we think a good candidate could be the IEEE 1394 RCP protocol (see [Sto03]).

Better understanding the general multi-clock framework. The two-clock example that we have presented page 114 is rather interesting as it presents a converging phenomenon (after each loop, in location ℓ_0, the value of clock y becomes closer and closer to 1). Together with Marcin Judziński (University of Warwick, England) and Thomas Brihaye, we are currently investigating further the probabilistic semantics, trying to detect the presence or absence of such convergence phenomenon. In particular we think that there cannot be any converging phenomenon if we do not bound the values of the clocks (the example mentioned above then really requires bounded intervals), and we claim the following:

☞ **Claim 6.17 (*ongoing work*)** *Assume that $\mathcal{A}$ is a timed automaton with arbitrary many clocks, and that for every configuration s of $\mathcal{A}$, $I(s)$ is unbounded. Assume furthermore exponential distributions over delays. Then the qualitative model-checking problem for ω-regular properties is decidable in* PSPACE.

The intuition behind this result is the following: from every configuration, the probability that the values of the clocks become larger than the maximal constant of the automaton is bounded from below, hence, in-the-long-run,

the values of the clocks will almost-surely become larger than the maximal constant. In the unbounded zone, the automaton behaves (from a probabilistic point-of-view) as a CTMC, because we somehow lose the memory of the past and become 'multiplicative'. Using such arguments, we conjecture that the same abstraction as in section 6.3 is correct for the qualitative model-checking of ω-regular properties.

Model with several players. One of our motivations was to develop a model that could express real-time constraints as well as probabilistic constraints, but also non-determinism. It is thus natural, now that we have presented a model *à la* Markov chains (following standard terminology we can call them $\frac{1}{2}$-player games), to develop models *à la* Markov decision processes (or $1\frac{1}{2}$-player games) and even $2\frac{1}{2}$-player games. This work is in collaboration with Vojtěch Forejt (PhD student at Masaryk University, Brno, Czech Republic).

We will briefly describe the framework and the results we expect from our preliminary investigations. Let $\mathcal{A} = (\mathsf{AP}, X, L, \ell_0, \mathsf{Goal}, E, \mathsf{Inv}, \mathcal{L})$ be a timed automaton. We assume that its set of locations L is partitioned into three subsets: $L_\bigcirc$, $L_\Box$, and $L_\Diamond$. A location in $L_\bigcirc$ will be probabilistic. A location in $L_\Box$ or $L_\Diamond$ will be non-probabilistic and will belong respectively to Player $\Box$ and Player $\Diamond$. As in classical timed games, the two players play according to strategies. A strategy $\lambda_\Box$ (respectively $\lambda_\Diamond$) for Player $\Box$ (respectively Player $\Diamond$) is a function that associates to every finite run ϱ ending in $L_\Box$ (respectively $L_\Diamond$) a pair $(t, e) \in \mathbb{R}_{\geqslant 0} \times E$ that describes the next move to be done after prefix ϱ. In probabilistic locations $L_\bigcirc$, we assume distributions over delays and edges are given (as done in Section 6.2). We write $S_\bigcirc$ (respectively $S_\Box$, $S_\Diamond$) for the set of configurations (ℓ, v) with $\ell \in L_\bigcirc$ (respectively $\ell \in L_\Box$, $\ell \in L_\Diamond$).

Fixing strategies for the two players, we will define a probability measure over sets of runs, as it has been done in section 6.2. However, strategies are arbitrary, and we need to refine the previous definition and use the memory of what has been done so far. Hence, we fix a finite run ϱ in $\mathcal{A}$ and assume that $s = \mathsf{last}(\varrho)$. We also fix a strategy profile $\Lambda = (\lambda_\Diamond, \lambda_\Box)$, where $\lambda_\Diamond$ (respectively $\lambda_\Box$) is a Player $\Diamond$ (respectively Player $\Box$) strategy. For a sequence of edges $(e_i)_{1 \leqslant i \leqslant n}$, we define the following probability measure after ϱ under

strategy profile Λ:[6]

$$\mathbb{P}_\Lambda(\pi(\varrho, e_1 \dots e_n)) = \begin{cases} \int_{t \in I(s,e_1)} p_{s+t}(e_1) \cdot \mathbb{P}_\Lambda(\pi(\varrho', e_2 \dots e_n))\, \mathrm{d}\mu_s(t) & \text{if } s \in S_\bigcirc \\ \mathbb{P}_\Lambda(\pi(\varrho', e_2 \dots e_n)) & \text{if } s \in S_\star \text{ and } (t, e_1) = \lambda_\star(\varrho) \\ 0 & \text{if } s \in S_\star \text{ and } (t, e) = \lambda_\star(\varrho) \text{ with } e \neq e_1 \end{cases}$$

where $\star \in \{\Box, \Diamond\}$ represents one of the two players, and $\varrho' = \varrho \xrightarrow{t,e_1}$.

***Remark* 6.18** Compared to classical stochastic games, the form of the strategies for the two players is rather simple (a single move is specified by the strategy). We could imagine having a more general definition for strategies, but at first, this is a reasonable definition. ⌟

The above model is called a $2\frac{1}{2}$-player timed game. If $L_\Box = \emptyset$ (respectively $L_\Diamond = \emptyset$), this is called an existential (respectively universal) $1\frac{1}{2}$-player timed game. If $L_\Box = L_\Diamond = \emptyset$, we recover the model we have studied in this chapter. All problems that we have considered can be extended to $2\frac{1}{2}$-player timed games (and hence to $1\frac{1}{2}$-player timed games) as follows. We fix an ω-regular property φ and a $2\frac{1}{2}$-player timed game $\mathcal{A}$. We also fix a constant $c \in [0,1] \cap \mathbb{Q}$ and a comparison operator $\sim \in \{<, \leqslant, =, \geqslant, >\}$. The threshold problem asks whether there exists a strategy $\lambda_\Diamond$ for Player $\Diamond$ such that for every strategy $\lambda_\Box$ for Player $\Box$, writing $\Lambda = (\lambda_\Diamond, \lambda_\Box)$, $\mathbb{P}_\Lambda(s_0 \models \varphi) \sim c$. If $c = 1$ and $\sim$ is $=$, then this corresponds to the almost-sure model-checking problem.

☞ **Claim 6.19 (*ongoing work*)** *The threshold problem for $2\frac{1}{2}$-player timed games and ω-regular properties is undecidable.*

☞ **Claim 6.20 (*ongoing work*)** *The almost-sure model-checking problem for existential $1\frac{1}{2}$-player timed games with a single clock and ω-regular properties is decidable.*

The first claim is now proven, and relies on the simulation of a two-counter machine, which uses ideas similar to those developed in chapter 5, but probabilities are used instead of costs to check that the operations have been made according to the rules of the two-counter machine. The proof of the second claim is not complete yet, and is rather complex. One of the reasons is that the set of configurations in a timed game is uncountable, and strategies may have very erratic forms.

[6] In section 6.2, the probability measure only depended on the current state, and not on the full history, hence $\mathbb{P}_\mathcal{A}$ was defined in a memoryless way.

The two results above are somehow extremal, and there is much room left in between to get interesting decidability results.

Can we compute more involve properties? The work on the quantitative analysis of ω-regular properties that has been presented in section 6.5 is just a first step towards the quantitative analysis of timed automata under the probabilistic semantics. Indeed, some restrictions are technical and not completely satisfactory, and this is somehow frustrating not to be able to relax the most technical ones. Note that this is not obvious as we already know that our method does not apply in general, even if we use exponential distributions over delays (an example is given in [BBBM08]).

So far, we have only considered some restricted qualitative and quantitative model-checking problems. Many other properties that measure the performance of systems are of interest, like the expected time, or the steady-state distribution, *etc.* Also, some logics have been considered in the framework of finite Markov chains and CTMCs, for instance PCTL [HJ94] or CSL [ASSB00, BHHK03], that allows to represent many such properties of interest, see [BHHK05]. It is often quite difficult to automatically verify such logics, and approximation algorithms need to be developed (it is the case of CSL over CTMCs [BHHK03]), but tools can however be implemented [HKMS03]. Developing further algorithms for the quantitative analysis of timed and probabilistic systems is for sure a challenging direction of research.

Chapter 7

Conclusion and perspectives

In this thesis, I have presented some of my recent contributions to the verification of timed automata, from the developments of algorithms and abstractions for the verification of the most basic properties in timed automata (chapter 3) to the definition of logics suitable for expressing and verifying properties with timing constraints (chapter 4), and to the development of models that take into account other quantitative aspects, like cost variables in chapter 5 or probabilities in chapter 6.

At the end of each chapter, I have already given some further developments for the given chapter. I will recall the most important ones, and give further perspectives, that I have not started investigating yet.

Data structures and algorithms for linear-time timed properties

As already said at the end of chapter 4, I am quite interested in the development of data structures and algorithms for fragments of linear-time timed temporal logics that have a (relatively) low theoretical complexity. Indeed, algorithms that we have designed are not really practical and a symbolic approach needs definitely be developed. We know that DBMs (as they do exist now) are not appropriate, but we also think that a DBM-like structure with a dynamic number of variables could be adequate. Also, we think that channel machines might be used as a data structure as well, as the channel allows to store an (*a priori*) unbounded amount of timing information.

The question of the implementation of those developments in a tool naturally arises (such developments would not really make sense without an implementation supporting the theory). I don't plan to implement a tool on my own, but that would be a great opportunity if the Uppaal team was interested in such developments, and as already done in the past for abstractions

(see chapter 3), I would enjoy collaborating with them on that topic.

Model-checking quantitative properties

Modelling (and verifying) quantitative properties in timed systems will for sure be one of my main streams of research in the next few years.

A more accurate model of energy consumption. In the original model of weighted timed automata, cost variables are non-decreasing, and a more realistic model is to also allow non-monotonic cost variables. However as many problems in the initial framework were already undecidable, one needs to somehow relax the problems we look at, and we propose to use a single cost variable in a global invariant for the system. We consider at first the problem of computing possible executions of the system along which the cost variable always lays within some fixed interval.

We have already proven some preliminary results in that framework (see the end of chapter 5, or [BFL$^+$08]), but the picture is far from being complete, and a whole set of problems are open. Note that an untimed model of systems that consumes and refills (at once) energy has been recently proposed in [AKY08], but the model is less general, and the problems they have looked at so far are different from ours.

Computing approximations. Another important research direction is the idea that the undecidability results that we have obtained either for the two-player framework or for the various model-checking problems heavily rely on the precision of the measures, hence this is most likely that we will *eg.* be able to design algorithms that computes approximations of the optimal cost in the two-player framework (even though several of our attempts have failed so far). However this is for sure a challenging research direction, because this is almost as interesting to know an approximate value of the optimal cost (if we know how far we are from the optimal cost) as to precisely know the optimal cost.

Simplified models for timing constraints. We have seen in chapter 5 many undecidability results. On the positive hand, we have also mentioned that some of the problems become decidable when we restrict to single-clock weighted timed automata. Furthermore the complexities are not that high, compared to classical complexities in the domain of real-time systems.

I think that this is worth pursuing this line of research, which consists in simplifying the way timing constraints are expressed. This restriction is

not new, and already basic properties can be verified more efficiently if we restrict to single-clock timed automata [LMS04]. Also, as mentioned in subsection 4.4.2, the emptiness of alternating timed automata is decidable only if we restrict to a single clock [LW08]. In a similar vein, the idea of using simplified models of time was mentioned in [Lar05], because it yielded drastic improvements in the theoretical complexities of many problems. I thus think that this is worth simplifying the way we express timing constraints so that it becomes possible to express other kinds of quantitative constraints. For instance, we could consider two-player games over single-clock weighted timed automata where the objective is to minimise/maximise the mean-cost. Also, [FL08] has proven that the corner-point abstraction is sound for the discounted-cost optimisation problem (in the single-player framework). This is an interesting measure, hence we should study the two-player framework, in particular for single-clock automata.

One could however say that one clock might not be sufficient to express properly the timing constraints in a real system. It might indeed be right, because most of the time, it is easier to model a system in a compositional manner, each component having its proper timing constraints (see the (simplified) car model in the introduction on page 7). A single clock for such a system is then probably not powerful enough, and at least one clock per component should be allowed. However, with one clock per component and accurate synchronisations, we can probably mimic all the undecidability proofs of chapter 5. A suggestion would then be to use a softer way for synchronising components. I have no precise suggestion yet, but [ABG+08] has proposed a very soft way of synchronising, based on clocks and not on events, and I think that it would be worth understanding whether this could help getting classes of weighted networks of single-clock timed automata that would be decidable and expressive enough for representing real-life systems.

Probabilistic analysis of timed automata. The model of timed automata with probabilities that we have presented in chapter 6 is rather recent, and only few problems have been investigated so far. We have obtained decidability and computability results only in a single-clock framework, but as we have said before, this is not problematic to have a simplified model for timing constraints if we can express other kinds of quantitative constraints. We thus plan to try to understand further our model, and as said at the end of chapter 6, we also investigate extensions of our model with non-determinism (and several players). We think that it may lead to an interesting model that allows, non-determinism, probabilistic choices, and timing constraints. We refer to the end of chapter 6 for more details.

From real-time models to reality.

Another important issue in the world of real-time systems concerns the adequacy of the timed-automata model with respect to real systems. I will briefly describe some of the issues.

Few years ago, we have proposed logics to express properties of systems where transient states are removed, and designed model-checking algorithms for those logics [BBBL05, BBBL06]. The motivation behind these works was the modelisation of PLCs (Programmable Logic Controllers) written in SFC (Sequential Function Chart), used in the area of industrial automation. Formal models that could be done of such controllers were not accurate, and the problem came from transient states that were visited but should not have been taken into account in the verification process.

The first original motivation for our work on timed automata and probabilities (chapter 6) was also the implementability paradigms, with the idea that unlikely events will actually not happen in a real system. One could use similar ideas to abstract transient states. For instance, one could abstract a property '$\mathbf{G}\,\varphi$' (φ should always hold) into the requirement that the probability of not satisfying φ along runs be less than 2% (or even 0%).

In an orthogonal way, as already said in the introduction, [DDR04] have proposed a notion of robustness that ensures the implementability of systems modelled as timed automata (on a simple model of processor). The corresponding robust model-checking problem has generated quite a lot of works [DDMR04, BMR06, DK06, Dim07, SF07, BMR08, DDMR08], with the development model-checking algorithms for rather large classes of properties, and the development of zone-based algorithms.

While investigating the robust model-checking of linear-time timed temporal logics [BMR08], we got the impression that we could develop algorithms based on channel machines for all the properties for which we can decide the robust model-checking. Indeed, so far, algorithms are really *ad-hoc* and dedicated to specific classes of properties, and the channel machine approach seems both elegant and unifying.

Bibliography

[AAM06] Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006.

[ABBL03] Luca Aceto, Patricia Bouyer, Augusto Burgueño, and Kim G. Larsen. The power of reachability testing for timed automata. *Theoretical Computer Science*, 300(1–3):411–475, 2003.

[ABG+08] S. Akshay, Benedikt Bollig, Paul Gastin, Madhavan Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. In *Proc. 19th International Conference on Concurrency Theory (CONCUR'08)*, volume 5201 of *Lecture Notes in Computer Science*, pages ??–?? Springer, 2008.

[ABM04] Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability in weighted timed games. In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.

[ACD90] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *Proc. 5th Annual Symposium on Logic in Computer Science (LICS'90)*, pages 414–425. IEEE Computer Society Press, 1990.

[ACD91] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for probabilistic real-time systems. In *Proc. 18th International Colloquium on Automata, Languages and Programming (ICALP'91)*, volume 510 of *Lecture Notes in Computer Science*, pages 115–126. Springer, 1991.

[ACD92] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Verifying automata specifications of probabilistic real-time systems. In *Proc. REX Workshop on Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 28–44. Springer, 1992.

[ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AFH91] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. In *Proc. 10th Annual ACM Symposium on Principles of Distributed Computing (PODC'91)*, pages 139–152. ACM, 1991.

[AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[AH89] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. In *Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*, pages 164–169. IEEE Computer Society Press, 1989.

[AH90] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proc. 5th Annual Symposium on Logic in Computer Science (LICS'90)*, pages 390–401. IEEE Computer Society Press, 1990.

[AH92a] Rajeev Alur and Thomas A. Henzinger. Back to the future: towards a theory of timed regular languages. In *Proc. 33rd Annual Symposium on Foundations of Computer Science (FOCS'92)*, pages 177–186. IEEE Computer Society Press, 1992.

[AH92b] Rajeev Alur and Thomas A. Henzinger. Logics and models of Real-Time: a survey. In *Proc. REX Workshop on Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer, 1992.

[AH93] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[AH94] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[AJ96] Parosh A. Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[AJ98] Parosh Aziz Abdulla and Bengt Jonsson. Verifying networks of timed processes. In *Proc. 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 1998.

[AKY08] Parosh Aziz Abdulla, Pavel Krčál, and Wang Yi. R-automata. In *Proc. 19th International Conference on Concurrency Theory (CONCUR'08)*, volume 5201 of *Lecture Notes in Computer Science*, pages 67–81. Springer, 2008.

[AL02] Luca Aceto and François Laroussinie. Is your model-checker on time ? on the complexity of model-checking for timed modal logics. *Journal of Logic and Algebraic Programming*, 52–53:7–51, 2002.

[ALM05] Rajeev Alur, Salvatore La Torre, and P. Madhusudan. Perturbed timed automata. In *Proc. 8th International Workshop on Hybrid Systems: Computation and Control (HSCC'05)*, volume 3414 of *Lecture Notes in Computer Science*, pages 70–85. Springer, 2005.

[ALP01] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2001.

[Alu91] Rajeev Alur. *Techniques for Automatic Verification of Real-Time Systems*. PhD thesis, Stanford University, Stanford, CA, USA, 1991.

[AM99] Eugene Asarin and Oded Maler. As soon as possible: Time optimal control for timed automata. In *Proc. 2nd International Workshop on Hybrid Systems: Computation and Control (HSCC'99)*, volume 1569 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.

[AM01] Yasmina Abdeddaim and Oded Maler. Job-shop scheduling using timed automata. In *Proc. 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 478–492. Springer, 2001.

[AMPS98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier Science, 1998.

[AN01] Parosh Aziz Abdulla and Aletta Nylén. Timed Petri nets and bqos. In *Proc. 22nd International Conference on Application and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2001.

[ASSB00] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert K. Brayton. Model-checking continuous-time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

[AT05] Karine Altisen and Stavros Tripakis. Implementation of timed automata: An issue of semantics or modeling? In *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2005.

[BBB+07] Christel Baier, Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Marcus Größer. Probabilistic and topological semantics for timed automata. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *Lecture Notes in Computer Science*, pages 179–191. Springer, 2007.

[BBB+08a] Christel Baier, Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Marcus Größer. Almost-sure model checking of infinite paths in one-clock timed automata. In *Proc. 23rd Annual Symposium on Logic in Computer Science (LICS'08)*, pages 217–226. IEEE Computer Society Press, 2008.

[BBB+08b] Christel Baier, Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Marcus Größer. A probabilistic semantics for timed automata. Research Report LSV-08-13, Laboratoire Spécification & Vérification, ENS de Cachan, France, 2008.

[BBBL05] Houda Bel Mokadem, Béatrice Bérard, Patricia Bouyer, and François Laroussinie. A new modality for almost everywhere properties in timed automata. In *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2005.

[BBBL06] Houda Bel Mokadem, Béatrice Bérard, Patricia Bouyer, and François Laroussinie. Timed temporal logics for abstracting transient states. In *Proc. 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2006.

[BBBM08] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Quantitative model-checking of one-clock timed automata under probabilistic semantics. In *Proc. 5th International Conference on Quantitative Evaluation of Systems (QEST'08)*. IEEE Computer Society Press, 2008.

[BBBR07] Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem. *Formal Methods in System Design*, 31(2):135–175, 2007.

[BBC06a] Patricia Bouyer, Laura Bozzelli, and Fabrice Chevalier. Controller synthesis for MTL specifications. In *Proc. 17th International Conference on Concurrency Theory (CONCUR'06)*, volume 4137 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2006.

[BBC06b] Patricia Bouyer, Thomas Brihaye, and Fabrice Chevalier. Control in o-minimal hybrid systems. In *Proc. 21st Annual Symposium on Logic in Computer Science (LICS'06)*, pages 367–378. IEEE Computer Society Press, 2006.

[BBC07] Patricia Bouyer, Thomas Brihaye, and Fabrice Chevalier. Weighted o-minimal hybrid systems are more decidable than weighted timed automata! In *Proc. Symposium on Logical Foundations of Computer Science (LFCS'07)*, volume 4514 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2007.

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim G. Larsen. Static guard analysis in timed automata verification. In *Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–277. Springer, 2003.

[BBJ+08] Patricia Bouyer, Thomas Brihaye, Marcin Jurdziński, Ranko Lazić, and Michał Rutkowski. Average-price and reachability-price games on hybrid automata with strong resets. In *Proc. 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, Lecture Notes in Computer Science. Springer, 2008.

[BBL04] Patricia Bouyer, Ed Brinksma, and Kim G. Larsen. Staying alive as cheaply as possible. In *Proc. 7th International Workshop on Hybrid Systems: Computation and Control (HSCC'04)*, volume 2993 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2004.

[BBL08] Patricia Bouyer, Ed Brinksma, and Kim G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):2–23, 2008.

[BBLP04] Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and upper bounds in zone based abstractions of timed automata. In *Proc. 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *Lecture Notes in Computer Science*, pages 312–326. Springer, 2004.

[BBLP05] Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelànek. Zone based abstractions for timed automata exploiting lower and upper bounds. *International Journal on Software Tools for Technology Transfer*, 8(3):204–215, 2005.

[BBM06] Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5):188–194, 2006.

[BBP04] Béatrice Bérard, Patricia Bouyer, and Antoine Petit. Analysing the PGM protocol with Uppaal. *International Journal of Production Research*, 42(14):2773–2791, 2004.

[BBR04] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. Model-checking for weighted timed automata. In *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, volume 3253 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2004.

[BBR05] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2005.

[BC05] Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 10(4):393–405, 2005.

[BCFL04] Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal strategies in priced timed game automata. In *Proc. 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, 2004.

[BCFL05] Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Synthesis of optimal strategies using HyTech. In *Proc. Workshop on Games in Design and Verification (GDV'04)*, volume 119(1) of *Electronic Notes in Theoretical Computer Science*, pages 11–31. Elsevier Science Publishers, 2005.

[BCM05] Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. In *Proc. 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 432–443. Springer, 2005.

[BCM08] Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. *Information and Computation*, 2008. To appear.

[BDFP04] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2–3):291–345, 2004.

[BDGP98] Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2–3):145–182, 1998.

[BDHK06] Henrik C. Bohnenkamp, Pedro R. DÁrgenio, Holger Hermanns, and Joost-Pieter Katoen. MoDeST: A compositional modeling formalism for hard and softly timed systems. *IEEE Transactions on Software Engineering*, 32(10):812–830, 1006.

[BDL+06] Gerd Behrmann, Alexandre David, Kim G. Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 125–126. IEEE Computer Society Press, 2006.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: a model-checking tool for real-time systems. In *Proc. 10th International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 1998.

[Ben02] Johan Bengtsson. *Clocks, DBMs ans States in Timed Systems*. PhD thesis, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2002.

[BFH+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

[BFL+08] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In *Proc. 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, Lecture Notes in Computer Science. Springer, 2008.

[BG02] Mario Bravetti and Roberto Gorrieri. The theory of interactive generalized semi-Markov processes. *Theoretical Computer Science*, 282(1):5–32, 2002.

[BHHK03] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(7):524–541, 2003.

[BHHK05] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model checking meets performance evaluation. *SIGMETRICS Performance Evaluation Review*, 32(4):10–15, 2005.

[BHK07] Henrik C. Bohnenkamp, Holger Hermanns, and Joost-Pieter Katoen. Motor: The MoDeST tool environment. In *Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *Lecture Notes in Computer Science*, pages 500–504. Springer, 2007.

[BHR06a] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Extended timed automata and time Petri nets. In *Proc. 6th International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 91–100. IEEE Computer Society Press, 2006.

[BHR06b] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed Petri nets and timed automata: On the discriminating power of Zeno sequences. In *Proc. 33rd International Colloquium on Automata, Languages and Programming (ICALP'06)*, volume 4052 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2006.

[BHR06c] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed unfoldings for networks of timed automata. In *Proc. 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 292–306. Springer, 2006.

[BHR08] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed Petri nets and timed automata: On the discriminating power of Zeno sequences. *Information and Computation*, 206(1):73–107, 2008.

[BLM07] Patricia Bouyer, Kim G. Larsen, and Nicolas Markey. Model-checking one-clock priced timed automata. In *Proc. 10th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'07)*, volume 4423 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2007.

[BLM08] Patricia Bouyer, Kim G. Larsen, and Nicolas Markey. Model checking one-clock priced timed automata. *Logical Methods in Computer Science*, 4(2:9), 2008.

[BLMR06] Patricia Bouyer, Kim G. Larsen, Nicolas Markey, and Jacob I. Rasmussen. Almost optimal strategies in one-clock priced timed automata. In *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.

[BLR04] Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen. Priced timed automata: Decidability results, algorithms, and applications. In *Proc. 3rd International Symposium on Formal Methods for Components and Objects (FMCO'04)*, volume 3657 of *Lecture Notes in Computer Science*, pages 162–186. Springer, 2004.

[BLR05a] Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen. Optimal scheduling using priced timed automata. *ACM Sigmetrics Performancs Evaluation Review*, 32(4):34–40, 2005.

[BLR05b] Patricia Bouyer, François Laroussinie, and Pierre-Alain Reynier. Diagonal constraints in timed automata: Forward analysis of timed systems. In *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2005.

[BM83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In *Proc. IFIP 9th World Computer Congress*, volume 83 of *Information Processing*, pages 41–46. North-Holland/ IFIP, 1983.

[BM07] Patricia Bouyer and Nicolas Markey. Costs are expensive! In *Proc. 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'07)*, volume 4763 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2007.

[BMO+08] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, Philippe Schnoebelen, and James B. Worrell. On termination for faulty channel machines. In *Proc. 25th Annual Symposium on Theoretical Aspects of Computer Science (STACS'08)*, pages 121–132, 2008.

[BMOW07] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. The cost of punctuality. In *Proc. 22nd Annual Symposium on Logic in Computer Science (LICS'07)*, pages 109–118. IEEE Computer Society Press, 2007.

[BMOW08] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. On expressiveness and complexity in real-time model checking. In *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*. Lecture Notes in Computer Science, 2008.

[BMR06] Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust model-checking of timed automata. In *Proc. 7th Latin American Symposium on Theoretical Informatics (LATIN'06)*, volume 3887 of *Lecture Notes in Computer Science*, pages 238–249. Springer, 2006.

[BMR08] Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust analysis of timed automata via channel machines. In *Proc. 11th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'08)*, volume 4962 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2008.

[Bou03] Patricia Bouyer. Untameable timed automata! In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631. Springer, 2003.

[Bou04] Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.

[Bré99] Pierre Brémaud. *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer, 1999.

[Bri07] Thomas Brihaye. Words and bisimulation of dynamical systems. *Discrete Mathematics and Theoretical Computer Science*, 9(2):11–31, 2007.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[CE82] Edmund M. Clarke and E. Allen Emersen. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[CES83] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Verification of finite state concurrent systems using temporal logic specifications: A practical approach. In *Proc. 10th ACM Symposium on Principles of Programming Languages (POPL'83)*, pages 117–126. ACM, 1983.

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[CFPI96] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.

[CGP99] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model-Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[Che07] Fabrice Chevalier. *Logiques pour les systèmes temporisés : contrôle et expressivité*. PhD thesis, École Normale Supérieure de Cachan, France, 2007.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.

[CS08] Pierre Chambart and Philippe Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Proc. 23rd Annual Symposium on Logic in Computer Science (LICS'08)*, pages 205–216. IEEE Computer Society Press, 2008.

[CY92] Costas Courcoubetis and Mihalis Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4):385–415, 1992.

[CY95] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[Dav05] Alexandre David. Merging DBMs efficiently. In *Proc. 17th Nordic Workshop on Programming Theory (NWPT'05)*, 2005. DIKU, University of Copenhagen.

[DDMR04] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robustness and implementability of timed automata. Technical Report 2004.30, Centre Fédéré en Vérification, Belgium, 2004.

[DDMR08] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robust safety of timed automata. *Formal Methods in System Design*, 2008. To appear.

[DDR04] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics: From timed models to timed implementations. In *Proc. 7th International Workshop on Hybrid Systems: Computation and Control (HSCC'04)*, volume 2993 of *Lecture Notes in Computer Science*, pages 296–310. Springer, 2004.

[DHGP04] Alexandre David, John Håkansson, Larsen Kim G., and Paul Pettersson. Minimal DBM subtraction. In *Proc. 16th Nordic Workshop on Programming Theory (NWPT'04)*, pages 17–20, 2004. Uppsala University IT Technical Report 2004-041.

[Dil90] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems (1989)*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1990.

[Dim07] Cătălin Dima. Dynamical properties of timed automata revisited. In *Proc. 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'07)*, volume 4763 of *Lecture Notes in Computer Science*, pages 130–146. Springer, 2007.

[DK05] Pedro R. D'Argenio and Joost-Pieter Katoen. A theory of stochastic systems part i: Stochastic automata. *Information and Computation*, 203(1):1–38, 2005.

[DK06] Conrado Daws and Piotr Kordy. Symbolic robustness analysis of timed automata. In *Proc. 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 143–155, 2006.

[DOTY96] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool KRONOS. In *Proc. Hybrid Systems III: Verification and Control (1995)*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1996.

[DP03] Josée Desharnais and Prakash Panangaden. Continuous stochastic logic characterizes bisimulation of continuous-time Markov processes. *Journal of Logic and Algebraic Programming*, 56:99–115, 2003.

[DP06] Deepak D'Souza and Pavithra Prabhakar. On the expressiveness of MTL with past operators. In *Proc. 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2006.

[DP07] Deepak D'Souza and Pavithra Prabhakar. On the expressiveness of MTL in the pointwise and continuous semantics. *International Journal on Software Tools for Technology Transfer*, 9(1):1–4, 2007.

[DRP07] Deepak D'Souza, M. Raj Mohan, and Pavithra Prabhakar. Eliminating past operators in Metric Temporal Logic. Manuscript, 2007.

[DT98] Conrado Daws and Stavros Tripakis. Model-checking of real-time reachability properties using abstractions. In *Proc. 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.

[Fel68] William Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, 1968.

[Fin94] Alain Finkel. Decidability of the termination problem for completely specified protocols. *Distributed Computing*, 7(3):129–135, 1994.

[FL08] Uli Fahrenberg and Kim G. Larsen. Discount-optimal infinite runs in priced timed automata. In *Proc. 10th International Workshop on Verification of Infinite-State Systems (INFINITY'08)*, 2008. To appear.

[FS01] Alain Finkel and Philippe Schnoebelen. Well structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[GHJ97] Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan. Robust timed automata. In *Proc. International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 331–345. Springer, 1997.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proc. 11th Annual Symposim on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society Press, 1996.

[Hen98] Thomas A. Henzinger. It's about time: Real-time logics reviewed. In *Proc. 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 439–454. Springer, 1998.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTech: A model-checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[HK99] Thomas A. Henzinger and Peter W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221:369–392, 1999.

[HKMS03] Holger Hermanns, Joost-Pieter Katoen, Joachim Meyer-Kayser, and Markus Siegle. A tool for model-checking Markov chains. *International Journal on Software Tools for Technology Transfer*, 4:153–172, 2003.

[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(1):94–124, 1998.

[HKV96] Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In *Proc. 7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 514–529. Springer, 1996.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model-checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[HR00] Thomas A. Henzinger and Jean-François Raskin. Robust undecidability of timed and hybrid systems. In *Proc. 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2000.

[HR04] Yoram Hirshfeld and Alexander Rabinovich. Logics for real time: Decidability and complexity. *Fundamenta Informaticae*, 62(1):1–28, 2004.

[HR05] Yoram Hirshfeld and Alexander Rabinovich. Timer formulas and decidable metric temporal logic. *Information and Computation*, 198(2):148–178, 2005.

[HR07] Yoram Hirshfeld and Alexander Rabinovich. Expressiveness of metric modalities for continuous time. *Logical Methods in Computer Science*, 3(1:3), 2007.

[HRS98] Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In *Proc. 25th International Colloquium on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 1998.

[HSLL97] Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using Uppaal. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 2–13. IEEE Computer Society Press, 1997.

[JRLD07] Jan J. Jessen, Jacob I. Rasmussen, Kim G. Larsen, and Alexandre David. Guided controller synthesis for climate controller using Uppaal Tiga. In *Proc. 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'07)*, volume 4763 of *Lecture Notes in Computer Science*, pages 227–240. Springer, 2007.

[JT07] Marcin Jurdziński and Ashutosh Trivedi. Reachability-time games on timed automata. In *Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*, volume 4596 of *Lecture Notes in Computer Science*, pages 838–849. Springer, 2007.

[JT08] Marcin Jurdziński and Ashutosh Trivedi. Average-time games. Submitted, 2008.

[Kam68] Johan A.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, Los Angeles, CA, USA, 1968.

[Kar78] Richard M. Karp. A characterization of the minimum mean-cycle in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978.

[KNP04] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 2.0: A tool for probabilistic model checking. In *Proc. 1st International Conference on Quantitative Evaluation of Systems (QEST'04)*, pages 322–323. IEEE Computer Society Press, 2004.

[KNSS00] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In *Proc. 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2000.

[KNSS02] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[Kur94] Robert P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, 1994.

[Lar05] François Laroussinie. *Model checking temporisé — Algorithmes efficaces et complexité*. Mémoire d'habilitation, Université Paris 7, Paris, France, 2005.

[LBB+01] Kim G. Larsen, Gerd Behrmann, Ed Brinksma, Angskar Fehnker, Thomas Hune, Paul Pettersson, and Judi Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *Proc. 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 493–505. Springer, 2001.

[LL98] François Laroussinie and Kim G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *Proc. IFIP Joint International Conference on Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pages 439–456. Kluwer Academic, 1998.

[LMM02] Salvatore La Torre, Supratik Mukhopadhyay, and Aniello Murano. Optimal-reachability and control for acyclic weighted timed automata. In *Proc. 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002)*, volume 223 of *IFIP Conference Proceedings*, pages 485–497. Kluwer, 2002.

[LMS04] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking timed automata with one or two clocks. In *Proc. 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 387–401. Springer, 2004.

[LPS00] Gerardo Lafferriere, George J. Pappas, and Shankar Sastry. O-minimal hybrid systems. *Mathematics of Control, Signals, and Systems*, 13(1):1–21, 2000.

[LPWY99] Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.

[LPZ85] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In *Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer, 1985.

[LR05] Kim G. Larsen and Jacob I. Rasmussen. Optimal conditional scheduling for multi-priced timed automata. In *Proc. 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 234–249. Springer, 2005.

[LW05] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. In *Proc. 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2005.

[LW08] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Transactions on Computational Logic*, 9(2:10), 2008.

[Mer74] Philip M. Merlin. *A Study of the Recoverability of Computing Systems.* PhD thesis, University of California, Irvine, CA, USA, 1974.

[MH84] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theoretical Computer Science*, 32:321–330, 1984.

[MSS88] David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proc. 3rd Annual Symposium on Logic in Computer Science (LICS'88)*, pages 422–427. IEEE Computer Society Press, 1988.

[NV07] Sumit Nain and Moshe Y. Vardi. Branching vs. linear time: Semantical perspective. In *Proc. 5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07)*, volume 4762 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2007.

[OW05] Joël Ouaknine and James Worrell. On the decidability of Metric Temporal Logic. In *Proc. 20th Annual Symposium on Logic in Computer Science (LICS'05)*, pages 188–197. IEEE Computer Society Press, 2005.

[OW06a] Joël Ouaknine and James Worrell. On metric temporal logic and faulty Turing machines. In *Proc. 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.

[OW06b] Joël Ouaknine and James Worrell. Safety metric temporal logic is fully decidable. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2006.

[OW07] Joël Ouaknine and James Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3(1:8), 2007.

[Oxt57] John C. Oxtoby. The Banach-Mazur game and Banach category theorem. *Annals of Mathematical Studies*, 39:159–163, 1957. Contributions to the Theory of Games, volume 3.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[Pur98] Anuj Puri. Dynamical properties of timed automata. In *Proc. 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, volume 1486 of *Lecture Notes in Computer Science*, pages 210–227. Springer, 1998.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[Rab08] Alexander Rabinovich. Complexity of metric temporal logic with counting. In *Proc. 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, Lecture Notes in Computer Science. Springer, 2008.

[Ram74] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, Massachussetts Institute of Technology, Cambridge, MA, USA, 1974.

[Ras99] Jean-François Raskin. *Logics, Automata and Classical Theories for Deciding Real-Time*. PhD thesis, University of Namur, Namur, Belgium, 1999.

[Ras05] Jean-François Raskin. *An Introduction to Hybrid Automata*, chapter Handbook of Networked and Embedded Control Systems, pages 491–518. Springer, 2005.

[Rey04] Mark Reynolds. The complexity of the temporal logic over the reals. Submitted, 2004.

[RLS06] Jacob I. Rasmussen, Kim G. Larsen, and K. Subramani. On using priced timed automata to achieve optimal scheduling. *Formal Methods in System Design*, 29(1):97–114, 2006.

[SC85] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[Sch02] Philippe Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.

[SF07] Mani Swaminathan and Martin Fränzle. A symbolic decision procedure for robust safety of timed systems. In *Proc. 14th International Symposium on Temporal Representation and Reasoning (TIME'07)*, page 192. IEEE Computer Society Press, 2007.

[Sto03] Mariëlle Stoelinga. Fun with FireWire: A comparative study of formal verification methods applied to the IEEE 1394 root contention protocol. *Formal Aspects of Computing*, 14(3):328–337, 2003.

[Thom02] Wolfgang Thomas. Infinite games and verification. In *Proc. 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 58–64. Springer, 2002. Invited Tutorial.

[Var85] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE Computer Society Press, 1985.

[Var96] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proc. Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1996.

[Var98] Moshe Y. Vardi. Sometimes and not never re-revisited: On branching versus linear time. In *Proc. 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1998.

[Var01] Moshe Y. Vardi. Branching *vs* linear time: Final showdown. In *Proc. 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001. Invited talk.

[VV06] Daniele Varacca and Hagen Völzer. Temporal logics and model checking for fairly correct systems. In *Proc. 21st Annual Symposium on Logic in Computer Science (LICS'06)*, pages 389–398. IEEE Computer Society Press, 2006.

[VW94] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[Wol00] Pierre Wolper. Constructing automata from temporal logic formulas: A tutorial. In *Proc. European Educational Forum: School on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 2000.

[WVS83] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *Proc. 24th Annual Symposium on Foundations of Computer Science (FOCS'83)*, pages 185–194. IEEE Computer Society Press, 1983.

[ZP96] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1–2):343–359, 1996.