

Verification of Embedded Systems

Algorithms and Complexity

Mémoire d'Habilitation à Diriger des Recherches

Nicolas Markey

April 2011

This thesis is to be defended on 8 April 2011 before a committee composed of

- Eugene Asarin (reviewer)
- Nicolas Halbwachs
- Thomas A. Henzinger (reviewer)
- François Laroussinie
- Kim G. Larsen
- Anca Muscholl
- Philippe Schnoebelen
- Wolfgang Thomas (reviewer)

This document reports on joint works with

- Patricia Bouyer-Decitre
- Romain Brenguier
- Thomas Brihaye
- Fabrice Chevalier
- Arnaud Da Costa
- Martin De Wulf
- Laurent Doyen
- Uli Fahrenberg
- François Laroussinie
- Kim G. Larsen
- Ghassan Oreiby
- Joël Ouaknine
- Jean-François Raskin
- Jacob Illum Rasmussen
- Pierre-Alain Reynier
- Philippe Schnoebelen
- Jiří Srba
- James Worrell

Contents

Introduction	5
1 Preliminaries on timed and open systems	11
1.1 Basics on model-checking	12
1.1.1 Transition systems and temporal logics	12
1.1.2 Behavioural equivalences	14
1.1.3 Model checking, satisfiability and equivalence checking	15
1.2 Modeling and verifying open systems	17
1.2.1 Game automata	17
1.2.2 Verifying controllability and bisimilarity	18
1.3 Modeling and verifying timed systems	19
1.3.1 Timed automata	19
1.3.2 Model checking timed properties and timed bisimilarity	21
1.3.3 Weighted timed automata	27
1.3.4 Timed games	29
2 Verification of timed systems	31
2.1 Expressiveness of temporal logics	32
2.1.1 Expressiveness of MITL	32
2.1.2 Expressiveness of TPTL and MTL	33
2.2 Real-time model-checking with punctuality	37
2.2.1 Decidable fragments of MTL	37
2.2.2 <i>coFlatMTL</i> in the timed-word semantics	41
2.2.3 <i>coFlatMTL</i> in the signal semantics	46
2.3 Robust model-checking	48
2.3.1 Implementability of timed automata	48
2.3.2 Robust verification of safety properties	50
2.3.3 Robust verification of LTL and <i>coFlatMTL</i>	53
2.4 Conclusions and future works	55
3 Quantitative verification beyond real-time	57
3.1 Model-checking and games on weighted timed automata	58
3.1.1 Weighted temporal logics	58
3.1.2 Weighted game automata	61
3.2 Weighted timed automata under <i>energy constraints</i>	64

3.2.1	Adding constraints on the observer value	64
3.2.2	Lower-bound constraints	65
3.2.3	Interval constraints	68
3.3	Conclusions and perspectives	70
4	Verification of concurrent games	73
4.1	Alternating-time temporal logic	73
4.1.1	Expressiveness issues	74
4.1.2	ATL model checking	75
4.2	Extending ATL with strategy contexts	78
4.2.1	Expressiveness of ATL with strategy contexts	79
4.2.2	Model checking algorithm for ATL* with strategy contexts	81
4.2.3	A note on strategy logic	85
4.3	<i>Really concurrent</i> timed game automata	85
4.3.1	Timed concurrent game structures	86
4.3.2	Properties of strategies in timed CGSs	87
4.3.3	Timed ATL model checking on timed CGSs	88
4.3.4	Ruling out Zeno strategies	89
4.3.5	Extensions of the model	90
4.4	Conclusions and future works	90
	Conclusions and perspectives	93
	Bibliography	95

Introduction

Verification of embedded systems

Automatic systems are ubiquitous, from widespread electronic appliances (telecommunication devices or automotive equipment) to critical systems (industrial plants, medical devices or rocketry). With the rapid development of electronics and wireless communications, those systems are becoming more and more complex, and bugs are everywhere, as exemplified every day by uncountably many bugs, from infinite loops (*e.g.* the *leap-year bug* turning all Zune MP3 devices off on 31 December 2008) to race conditions (one of a sequence of problems leading to the North-East blackout on 14 August 2003) or arithmetic overflow (causing the burst of the Ariane 5 rocket on 4 June 1996).

This advocates for the development of formal methods for checking that reactive systems behave as expected. Mathematical techniques allow for possibly exhaustive exploration and rigorous diagnosis of the behaviours of the system under study. Different such techniques have emerged, among which model-based testing, proof techniques, and model checking. My research over the last ten years (including my Ph.D. thesis) has focused on the latter: model checking aims at *automatically* and *exhaustively* verifying that (a model of) the system under study satisfies (a formula expressing) some correctness property. In the most basic setting, the system is modelled as a finite-state automaton, while the property is expressed in temporal logics (logics extended with *temporal modalities*, *e.g.* expressing that some property will hold until another property becomes true). The model-checking process is schematically depicted on Figure 1.

Verification by model checking was defined in the late 1970's [Pnu77, CE82, QS82]. It has now been applied successfully on many industrial cases. As a proof of its importance and success, the *fathers* of model checking have won two Turing awards (Pnueli in 1996 for “*introducing temporal logics in computing science*”, especially in model checking; Clarke, Emerson and Sifakis in 2007 for making model checking “*a highly effective verification technology*”).

The main objective of research on model checking is to define and study a wide range of modelling formalisms, in order to understand their characteristics and their relative pros and cons. This is two-fold: first, a precise understanding of the relative expressiveness of these modelling languages (both theoretically and practically) allows one to decide in which case a given formalism is best suited, depending on the particular aspects one wants to check. Second, the design of model-checking algorithms is the central point in the area; making them as efficient as possible (again, both theoretically

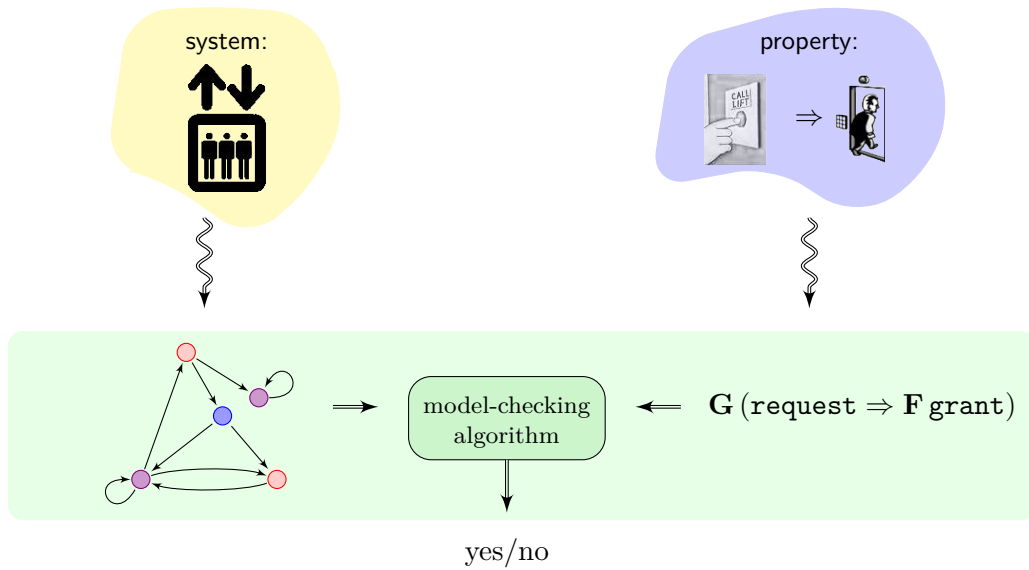


Figure 1: Model checking

and practically) is central to understand this technique and promote it in industrial applications.

My research interests are about both aspects in two specific settings, which are particularly relevant for the analysis of embedded systems (“*computer system designed to perform one or a few dedicated functions, often with real-time computing constraints*”, as defined by the Wikipedia): timed automata and concurrent-game automata.

The framework of timed automata extends classical finite-state automata with real-time constraints. The availability of transitions in the resulting *timed automata* depend on time, so that the structure of the automaton *evolves* during the execution. This provides an important and useful gain in expressiveness, while preserving decidability of several model-checking-related problems. Over the last ten years, timed automata have been extended to model other quantitative aspects, for instance energy consumption. Temporal logics have been extended accordingly, in order to express and verify quantitative properties.

Concurrent-game automata are another extension of finite-state automata, to a multi-agent setting: this provides a way of modelling interactions between the system under study and its environment (which may be composed of several other systems). In this case, the objective of model-checking is not to ensure that all behaviours are correct, but that it is possible to *control* the system in such a way that it behaves correctly, irrespective of what the environment does. There is a tight link between this and game theory: controlling the system corresponds to finding a winning strategy in a game against the environment. Temporal logics have been extended to express *controllability* properties, *e.g.*, that there is a strategy for avoiding a bad configuration of the system.

Content of this thesis

This document summarizes most of my research results in model checking timed and concurrent-game automata. It is organized as follows:

Chapter 1 is a more precise introduction to model checking, reporting the important definitions and basic results in the settings of timed automata and concurrent-game automata. This will settle the framework for the rest of the document.

Chapter 2 details my contributions on model checking in the setting of timed automata. The first part of this chapter considers the classical model of timed automata, and focuses on several timed temporal logics:

- on their expressive power: in a work with Patricia Bouyer-Decitre and Fabrice Chevalier in 2005, we solved the 15-year-old conjecture that the timed temporal logic MTL (where temporal modalities such as “until” are decorated with timing constraints) is strictly less expressive than TPTL (where timing constraints are enforced with explicit clocks in the formula).
- on model-checking algorithms: with Patricia Bouyer-Decitre, Joël Ouaknine and James B. Worrell, we introduced a new timed temporal logic, allowing punctual timing constraints while remaining decidable. Since several results by Rajeev Alur and Thomas Henzinger in the early 1990’s, it was believed that punctual constraints generally came with undecidability (in the setting of linear-time temporal logics).

In the second half of this chapter, we consider a variant of the semantics of timed automata. Indeed, the semantics of timed automata assumes infinite frequency of the hardware on which the automaton is executed: transitions are immediate, synchronization is perfect, clocks have infinite precision. This is to avoid imposing any bound on the speed or precision of the hardware, and is quite convenient for mathematical reasoning about those systems. However, this is unrealistic, and leads (among others) to the well-known *Zeno* phenomenon, where infinitely many transitions can take place in a finite delay. In order to prevent this kind of fake behaviours, Martin De Wulf, Laurent Doyen and Jean-François Raskin introduced a *parametrised semantics* in which the frequency of the CPU and the precision of clocks are parameters. During my postdoc in Jean-François Raskin’s lab, and then later in collaboration with Patricia Bouyer-Decitre and Pierre-Alain Reynier, I studied the problem of deciding the existence of sufficiently precise hardware on which a timed automaton can be implemented while preserving correctness properties proved on the model.

Chapter 3 studies an extension of timed automata with extra variables whose evolution is not constant in time (we call them *hybrid variables* in the sequel). It has been observed for a long time that time was not the only quantity of interest in model checking. Measuring the consumption of some resource may also be of interest. Unfortunately, most (even basic) extensions of timed automata with hybrid variables (*hybrid automata*) come with undecidable reachability problem. In 2001, timed automata with hybrid observers (a.k.a. weighted timed automata) have been introduced simultaneously by Alur

et al. and by Larsen *et al.*. In these automata, hybrid variables are *observers* and have no influence on the behaviour of the underlying timed automaton. The authors proved that *optimal reachability* is decidable in these models. In several papers with Patricia Bouyer-Decitre, Thomas Brihaye, Kim G. Larsen and Jacob Illum Rasmussen, we pursued the study of these models, proving undecidability of verifying weight-constrained properties, and decidability in restricted cases.

In 2008, with Patricia Bouyer-Decitre, Uli Fahrenberg, Kim G. Larsen and Jiří Srba, we introduced a different semantics for the same models (with positive and negative weights), which we called *energy constraints*. Contrary to the previous case, the first aim is not optimization but *management* of a resource, with the aim of maintaining the value of an observer variable within given bounds¹. This problem actually originated from one of our industrial partners in the European project QUASIMODO, who wanted to control the level of oil in a reservoir of a plastic injection moulding machine, ensuring that it never overflows or runs dry. While this semantics is one step closer to hybrid automata (since hybrid variables interfere, in a weak sense, with the behaviour of the underlying timed automaton), we were able to obtain some decidability results for this semantics, which will be presented in the second part of this chapter on weighted timed automata.

Chapter 4 deals with concurrent-game automata. These models were introduced in 2002 by Rajeev Alur, Thomas Henzinger and Orna Kupferman together with a new temporal logic, ATL, for expressing controllability properties on game models. With François Laroussinie and Ghassan Oreiby, we revisited the expressiveness and model-checking problem of this logic in the classical setting. With François Laroussinie and Arnaud Da Costa-Lopes, we proposed an alternative semantics for ATL, which is much more powerful as it can express non-zero-sum properties in concurrent games. We proved decidability of the model-checking problem for this logic.

In the second part of this chapter, in collaboration with Thomas Brihaye, François Laroussinie and Ghassan Oreiby, we extended concurrent-game automata with timing constraints. While this requires much technical work, our extension is more expressive than other existing models of timed games, while preserving similar decidability properties. In particular, we prove that a timed extension of ATL* is decidable on our models.

Other contributions not in this thesis

Several papers are not mentioned in this thesis, because they lie aside from the main streams of my research work. I briefly list these results below.

Path model checking. I introduced the problem of *path model checking* with Philippe Schnoebelen in 2003 [MS03]. This is a restriction of the model-checking problem to lasso-shaped models, with the aim of benefiting from the simplified model to obtain more efficient algorithms. Path model checking has several applications *e.g.* in intrusion detection (analysis of log files) or approximate model checking for Markov chains (a Monte-Carlo method for model checking probabilistic systems).

¹A similar problem was studied in [CdAHS03] in a different setting.

We proved that it is indeed possible to design specialized algorithms for this problem, with much improved complexity results [MS03]: path model checking can be achieved in polynomial time for very succinct temporal logics such as extensions of LTL². First-order-logic path model checking can be achieved in polynomial space, while it is non-elementary on finite-state automata. We extended this study to the μ -calculus, proving that path model checking was not easier than the general case [MS06], and to compressed words (which is useful for the analysis of log files), proving that LTL properties could be checked in polynomial time (in the length of the compressed word) [MS04a]. With Jean-François Raskin, we extended this study to the timed setting, considering different types of inputs: single timed paths, region paths and zone paths [MR04, MR06]. Again in this setting, we managed to obtain specialized algorithms that are more efficient than the existent algorithms on timed automata (for MITL and TCTL). We also obtained that model checking TPTL and MTL is decidable along finite zone-paths, while ultimately-periodic region paths are sufficient to make it undecidable.

Computing optimal winning coalitions. While Thomas Brihaye was a post-doctoral researcher at LSV, we worked with two students on the problem of computing the size of the smallest winning coalition(s) for reachability objectives in n -player concurrent games. We proved that deciding whether this size is less than a given value is NP-complete (by reduction of the SAT problem), and checking equality is DP-complete. We also studied various related problems [BGMR08].

Applications of game theory in telecommunications. Between 2006 and 2009, I was responsible of a small-size multi-disciplinary project with Marc Jungers, from the control laboratory SATIE of ENS Cachan. The aim of the project was to apply model-checking techniques for verifying power allocation policies in wireless telecommunication protocols. We used UppAal TIGA to check whether a given policy would allow correct bandwidth sharing between the users. Our results were published in [BJL⁺10].

Complexity of computing radical sums. Recently, the *square-root-sum* problem has been proven polynomial-time reducible to the different problems over stochastic games or recursive Markov chains [EY05, BBFK06, EY07]. The *square-root-sum* problem is to decide whether a sum of square roots of a set of integers is less than a given threshold. It is known to lie within PSPACE, but it is an open question whether it is in NP or even in PTIME. During a visit of Joël Ouaknine and James B. Worrell at LSV, we decided to have a look at the square-root-sum problems. In the end, we got new results for a different problem, namely the problem of deciding whether a sum of rational powers of rational numbers is zero, which we prove can be decided in TC⁰ (a circuit-complexity class within PTIME, see Table 3 on page 77) [HBM⁺10].

²Recently, LTL path model checking has been proven to be in AC¹(LOGDCFL) [KF09], which is a circuit-complexity class within PTIME.

Some notes about this thesis

In this document, my results are highlighted with a grey background. The document also contains four one-page explanations or reminders on different topics (expressiveness proof techniques on page 34, channel automata on page 43, complexity classes below and above PTIME on page 77 and alternating tree automata on page 82).

In the bibliography, references to my papers have hypertext links to the corresponding paper. For my complete list of publication, please visit

<http://www.lsv.ens-cachan.fr/~markey/publis.php>

1

Preliminaries on timed and open systems

Contents

1.1	Basics on model-checking	12
1.1.1	Transition systems and temporal logics	12
1.1.2	Behavioural equivalences	14
1.1.3	Model checking, satisfiability and equivalence checking	15
1.2	Modeling and verifying open systems	17
1.2.1	Game automata	17
1.2.2	Verifying controllability and bisimilarity	18
1.3	Modeling and verifying timed systems	19
1.3.1	Timed automata	19
1.3.2	Model checking timed properties and timed bisimilarity	21
1.3.3	Weighted timed automata	27
1.3.4	Timed games	29

This section briefly surveys the setting of model checking for timed and open systems. Its main purpose is to introduce the main formalisms that will be discussed in the rest of the document.

1.1 Basics on model-checking

This section introduces very common notions, and is there basically to fix some notations. We fix once and for all a finite set AP of atomic propositions.

1.1.1 Transition systems and temporal logics

Model checking is a formal method for automatically checking whether a system satisfies a given property. The most basic model for representing reactive systems was that of *finite-state labelled transition systems*. A *labelled transition system* is a 3-tuple $\mathcal{S} = \langle S, T, \text{label} \rangle$ where S is the set of states of \mathcal{S} , $T \subseteq S \times S$ is its set of transitions, and $\text{label}: S \rightarrow 2^{\text{AP}}$. Given a state $s \in S$, we write sT for the set of transitions issued from s .

Let $s \in S$. A *run* from s in \mathcal{S} is a (finite or infinite) word over S with first letter s , and such that any two consecutive letters form a transition in T . Any run w inherits the notions of length $|w|$, prefix $w_{\leq i}$, suffix $w_{\geq i}$, first state $\text{first}(w)$, last state $\text{last}(w)$ and i -th state w_i from the same notions on words. A run is *maximal* if it is infinite or his last state s is such that $sT = \emptyset$. We write $\text{Runs}_{\mathcal{S}}(s)$ for the set of runs of \mathcal{S} , and $\text{Runs}_{\mathcal{S}}^m(s)$ for the set of maximal runs from s . Given a run $w = (s_i)_{i \in I}$, the *trace* of w is the word over 2^{AP} defined as $\text{label}(w) = (\text{label}(s_i))_{i \in I}$ (which we abusively write $\text{label}(w)$).

The computation tree of \mathcal{S} from s is the tree $\mathcal{T} = \text{Runs}_{\mathcal{S}}(s)$. The one-letter word s is called the *root* of the tree¹, and denoted with $\text{root}(\mathcal{T})$. A *branch* of \mathcal{T} is a maximal run in \mathcal{T} . We write $\text{br}(\mathcal{T})$ for the set of branches of \mathcal{T} . We write $\text{Tree}_{\mathcal{S}}(s)$ for the computation tree of \mathcal{S} from s . Given such a tree and one of its nodes w , the subtree rooted at w is the tree $\text{Tree}_{\mathcal{S}}(\text{last}(w))$.

The most widely used formalisms for expressing properties for model checking is temporal logics. Temporal logics extend boolean logic with *modalities* used to express that something must occur after something else. Temporal logics have been studied by philosophers starting from the 1950's [Pri67, Pri68], and have been introduced in computer science in the late 1970's for model checking [Pnu77, CE82, QS82].

The *full computation tree logic* CTL* [EH83], which encompasses the main temporal logics, has the following grammar:

$$\begin{aligned} \text{CTL}^* \ni \varphi_s &::= p \mid \neg \varphi_s \mid \varphi_s \vee \varphi_s \mid \mathbf{E}\varphi_p \mid \mathbf{A}\varphi_p \\ \varphi_p &::= \varphi_s \mid \neg \varphi_p \mid \varphi_p \vee \varphi_p \mid \mathbf{X}\varphi_p \mid \varphi_p \mathbf{U} \varphi_p \end{aligned}$$

where p ranges over $\text{AP} \cup \{\mathbf{true}\}$.

¹Formally, this is not a tree as ε does not belong to it. It can easily be made a tree by dropping the letter s at the beginning of each word in $\text{Runs}_{\mathcal{S}}(s)$.

CTL* state formulas are evaluated on execution trees, while path formulas are evaluated on runs. The semantics is defined inductively on the structure of the formula:

$$\begin{array}{lll}
\mathcal{T} \models p & \Leftrightarrow & p \in \text{label}(\text{root}(\mathcal{T})) \\
\mathcal{T} \models \neg \varphi_s & \Leftrightarrow & \mathcal{T} \not\models \varphi_s \\
\mathcal{T} \models \varphi_s \vee \psi_s & \Leftrightarrow & \mathcal{T} \models \varphi_s \text{ or } \mathcal{T} \models \psi_s \\
\mathcal{T} \models \mathbf{E}\varphi_p & \Leftrightarrow & \exists w \in \text{br}(\mathcal{T}). w \models \varphi_p \\
\mathcal{T} \models \mathbf{A}\varphi_p & \Leftrightarrow & \forall w \in \text{br}(\mathcal{T}). w \models \varphi_p \\
w \models \varphi_s & \Leftrightarrow & \text{Tree}_{\mathcal{S}}(\text{first}(w)) \models \varphi_s \\
w \models \neg \varphi_p & \Leftrightarrow & w \not\models \varphi_p \\
w \models \varphi_p \vee \psi_p & \Leftrightarrow & w \models \varphi_p \text{ or } w \models \psi_p \\
w \models \mathbf{X}\varphi_p & \Leftrightarrow & w_{\geq 1} \models \varphi_p \\
w \models \varphi_p \mathbf{U} \psi_p & \Leftrightarrow & \exists j \geq 0. w_{\geq j} \models \psi_p \text{ and } \forall 0 \leq i < j. w_{\geq i} \models \varphi_p
\end{array}$$

Given a transition system \mathcal{S} , a state s and a formula φ of CTL*, we write $\mathcal{S}, s \models \varphi$ whenever $\text{Tree}_{\mathcal{S}}(s) \models \varphi$.

Useful shorthands are defined on top of CTL*: these include the classical boolean implication \Rightarrow and equivalence \Leftrightarrow , as well as the new modalities $\mathbf{F}\varphi_p$ (read “*eventually* φ_p ” and defined as $\text{true} \mathbf{U} \varphi_p$) and $\mathbf{G}\varphi_p$ (“*always* φ_p ”, defined as $\neg \mathbf{F} \neg \varphi_p$). Such modalities can be combined to express, for instance, that some property ψ will be true infinitely many times ($\mathbf{G}\mathbf{F}\psi$, which is often written as $\tilde{\mathbf{F}}\psi$) or, symmetrically, only finitely many times ($\mathbf{F}\mathbf{G}\neg\psi$, written $\tilde{\mathbf{G}}\neg\psi$).

Several fragments of CTL* are also classically defined: the *computation tree logic* CTL [CE82, QS82] is the restriction where path formulas are restricted to the following grammars:

$$\varphi_p ::= \neg \varphi_p \mid \mathbf{X}\varphi_s \mid \varphi_s \mathbf{U} \varphi_s$$

In particular, each modality (\mathbf{X} , \mathbf{U} , \mathbf{F} or \mathbf{G}) must be preceded with a path quantifier (\mathbf{E} or \mathbf{A}), so that CTL syntax is sometimes given as

$$\text{CTL} \ni \varphi ::= p \mid \neg \varphi \mid \varphi \vee \varphi \mid \mathbf{E}\mathbf{X}\varphi \mid \mathbf{A}\mathbf{X}\varphi \mid \mathbf{E}\varphi \mathbf{U} \varphi \mid \mathbf{A}\varphi \mathbf{U} \varphi$$

Additionally, it can be checked that

$$\neg(\mathbf{A}\varphi \mathbf{U} \psi) \equiv \mathbf{E}\mathbf{G} \neg \psi \vee \mathbf{E}(\neg \psi) \mathbf{U} (\neg \varphi \wedge \neg \psi) \quad (1.1)$$

(where equivalence in this equation means that both formulas have the same truth value on any run of any transition system) so that CTL is often defined as

$$\text{CTL} \ni \varphi ::= p \mid \neg \varphi \mid \varphi \vee \varphi \mid \mathbf{E}\mathbf{X}\varphi \mid \mathbf{E}\mathbf{G}\varphi \mid \mathbf{E}\varphi \mathbf{U} \varphi.$$

The *linear-time temporal logic* LTL [Pnu77] is the fragment of CTL* defined by

$$\text{LTL} \ni \varphi_p ::= p \mid \neg \varphi_p \mid \varphi_p \vee \varphi_p \mid \mathbf{X}\varphi_p \mid \varphi_p \mathbf{U} \varphi_p$$

In order to be consistent with the above definitions and notations, we define $\mathcal{S}, s \models \varphi$, for $\varphi \in \text{LTL}$, as being equivalent to $\mathcal{S}, s \models \mathbf{A}\varphi$ (where $\mathbf{A}\varphi$ is a CTL^* formula).

Much work has been devoted to the study of expressiveness and algorithms for these logics. Regarding expressiveness, LTL and CTL form incomparable fragments of CTL^* : LTL is equivalent to monadic first-order logic on words, while CTL (and CTL^*) is a fragment of the monadic second-order logic over trees. Many extensions of these logics have been proposed and studied: past-time modalities [LS94, KP95, LMS02b], automata-based modalities [Wol83], fixpoint operators [Koz83, Var88], regular expressions [AFF⁺02, Lan07], ...

1.1.2 Behavioural equivalences

Behavioural equivalences are relations between states of a transition system. They provide a way of saying when two states are *equivalent*. In the same way as there are several possible temporal logics, there exist several behavioural equivalence. We will consider two:

- *trace equivalence* [Hoa80]: a relation $I \subseteq S \times S$ is a *trace-inclusion* relation if, for any $(s, s') \in I$ and any run w from s , there is a run w' from s' such that $\text{label}(w) = \text{label}(w')$. A relation E such that E and E^{-1} are trace-inclusion relations is a *trace-equivalence* relation. Two states whose pair is in a trace-equivalence relation are said *trace equivalent*.
- *bisimulation* [Mil80]: a relation $M \subseteq S \times S$ is a *simulation* relation if, for any $(s, s') \in M$, $\text{label}(s) = \text{label}(s')$ and for any transition $(s, t) \in T$ from s , there is a transition $(s', t') \in T$ from s' such that $(t, t') \in M$. A relation B such that B and B^{-1} are simulation relations is called a *bisimulation*, and two related elements are said bisimilar.

Obviously, bisimilarity implies trace equivalence. Moreover, these relations have very tight links with classical temporal logics. In particular, the following results have been proved:

Theorem ([HM85, BCG88]). • *Two states are trace-equivalent if, and only if, they satisfy the same set of LTL;*

- *Two states of a finitely-branching transition system are bisimilar if, and only if, they satisfy the same set of CTL (or CTL^*) formulas.*

For infinitely-branching transition systems, only one implication holds: bisimilar states satisfy the same set of CTL formulas. Figure 1.1 displays an example where the converse does not hold: s_1 and s_2 satisfy the same CTL formulas (can be proved by induction on the structure of the formula), but are not bisimilar (because of the unmatched transition from s_1 to s_2).

These properties will reveal helpful for model checking, since they provide a way of verifying whether a formula holds on a transition system by checking it on another transition system.

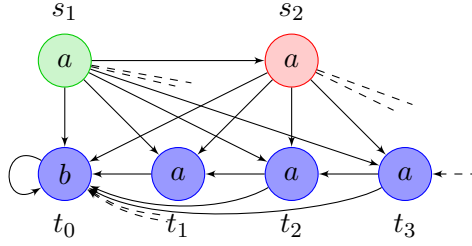


Figure 1.1: A (supposedly) infinitely-branching transition system

1.1.3 Model checking, satisfiability and equivalence checking

We are now in a position to formally define the *model-checking* problem, which is central in this document:

Definition. *The model-checking problem is the following decision problem:*

Problem: *Model checking*

Input: *a transition system \mathcal{S} , a state s of \mathcal{S} , a formula φ in some temporal logic;*

Question: *Does $\mathcal{S}, s \models \varphi$?*

We will also be interested in *satisfiability*:

Definition. *The satisfiability problem is the following decision problem:*

Problem: *Satisfiability*

Input: *A formula φ in some temporal logic, a class of transition systems \mathcal{C} ²;*

Question: *Do there exist \mathcal{S} in \mathcal{C} and a state s of \mathcal{S} such that $\mathcal{S}, s \models \varphi$?*

Both problems has been much studied for all three temporal logics we have defined above: first, LTL can be translated to (alternating) Büchi automata, which provides an algorithm for model checking [VW86, Var95]. The states of the automaton are consistent sets of subformulas of the LTL formula under study (consistency also imposing that a state containing $\psi_1 \mathbf{U} \psi_2$ also contains either ψ_1 or ψ_2 and $\mathbf{X}(\psi_1 \mathbf{U} \psi_2)$), and the transitions enforce consistency between consecutive states, by enforcing that a state containing a subformula $\mathbf{X}\psi$ is followed by states containing ψ . In the end, the automaton has exponential size, but non-emptiness (of the product with the transition system under study) can be checked on-the-fly in polynomial space. This provides algorithms for both satisfiability and for model checking (by considering the product of this automaton with the transition system under checking).

Concerning CTL and CTL*, the conceptually simplest algorithms consist in labelling states of the transition system by the state-subformulas they satisfy. It then amounts to deciding whether a state satisfies an LTL formula (for CTL*) or an LTL formula made of just one temporal modality (for CTL). The former can be achieved using the LTL

²This part of the input is often implicit.

model-checking algorithm. The second one can be achieved more efficiently by a fixpoint computation (on the set of states of the transition system): for instance, the set of states satisfying $\varphi = \mathbf{E}a \mathbf{U} b$ (which we write $\llbracket \varphi \rrbracket$) is the least fixpoint of the mapping

$$f: 2^S \rightarrow 2^S$$

$$X \mapsto \left\{ s \in S \mid \exists t \in S. (s, t) \in T \text{ and } \left[b \in \text{label}(t) \text{ or } a \in \text{label}(t) \text{ and } t \in X \right] \right\}.$$

As f is non-decreasing, this fixpoint exists and can be computed iteratively in polynomial time: writing $\text{Pre}(X) = \{s \in S \mid \exists t \in X. (s, t) \in T\}$ for the set of predecessors of X , it suffices to compute the sequence defined by $X_0 = \emptyset$ and $X_{i+1} = X_i \cup \text{Pre}(\llbracket a \rrbracket \cup \llbracket b \rrbracket \cap X_i)$ until the least fixpoint is reached. Similar procedure can be obtained for other modalities of CTL, which provides us with a polynomial-time algorithm.

Another approach, based on alternating tree automata, have been proposed for both CTL and CTL* [EJ99, KVV00]. This provides an automata-theoretic solution to both model checking and satisfiability for CTL and CTL*. To sum up:

Theorem ([CE82, QS82, SC85, CES86]). • *CTL model checking is PTIME-complete, and satisfiability is EXPTIME-complete.*

- *LTL model checking is PSPACE-complete, as well as satisfiability.*
- *CTL* model checking is PSPACE-complete, and satisfiability is 2-EXPTIME-complete.*

Behavioural-equivalence checking can also be interesting in many cases: it consists in deciding whether two states of a transition system are trace-equivalent or bisimilar (or linked by any other behavioural equivalence). Trace-equivalence is reducible and co-reducible to the problem of language equivalence of two finite-state automata, which is well-known to be PSPACE-complete [GJ79].

Bisimulation checking can be achieved computing the largest bisimulation relation as the largest fixpoint of the following function:

$$f: 2^{S \times S} \rightarrow 2^{S \times S}$$

$$R \mapsto \left\{ (s, s') \mid \begin{array}{l} \text{label}(s) = \text{label}(s') \text{ and} \\ \forall (s, t) \in T. \exists t'. (s', t') \in T \text{ and } (t, t') \in R \text{ and} \\ \forall (s', t') \in T. \exists t. (s, t) \in T \text{ and } (t, t') \in R \end{array} \right\}$$

We have:

Theorem ([KS83, SJ01]). • *Trace-equivalence checking is PSPACE-complete.*

- *Bisimulation checking is PTIME-complete.*

Another approach to deciding bisimilarity is in terms of two-player games (see Section 1.2.1). The game is played in the product $S \times S$, where Player 1, trying to prove non-bisimilarity, challenges Player 2 by proposing a transition from one of the two supposedly bisimilar states. Player 2 must propose a matching move from the other state, so that both target states have the same labelling. Player 2 has a winning strategy if, and only if, the initial two states are bisimilar. As will be shown below, this can be decided in polynomial time.

1.2 Modeling and verifying open systems

Embedded systems most often interact with their environment: they receive input from the external world, and must react appropriately in order to meet their specification. In that setting, the problem is to find the appropriate reaction to external stimuli, *i.e.*, to compute a winning strategy in a two-player game against the environment. This problem obviously extends to the multi-agent setting, which we present here.

1.2.1 Game automata

In order to represent these interactions, transition systems have been extended to a multi-agent setting: the evolution of the system depends on the (simultaneous) choices of the agents at each step. Formally, a *concurrent game structures* (CGS) [AHK02] is a 5-tuple $\mathcal{G} = \langle \mathcal{S}, \text{Agt}, \text{Mov}, \text{Chc}, \text{Tab} \rangle$ where \mathcal{S} is the underlying transition system, Agt is a finite set of agents, Mov is the set of actions players can play, $\text{Chc}: \mathcal{S} \times \text{Agt} \rightarrow 2^{\text{Mov}} \setminus \emptyset$ indicates which actions are available to each agent at each state, and $\text{Tab}: \mathcal{S} \times \text{Mov}^{\text{Agt}} \rightarrow \mathcal{T}$ indicates the transition to be taken given the moves chosen by the agents.

Figure 1.2 displays a concurrent-game automaton encoding the well-known “*paper-rock-scissors*” game: both players simultaneously select one move among *paper*, *rock* and *scissors*, with *paper* winning over *rock*, *rock* over *scissors* and *scissors* over *paper*. In case of a draw, a new round is played. Other examples of concurrent games include in

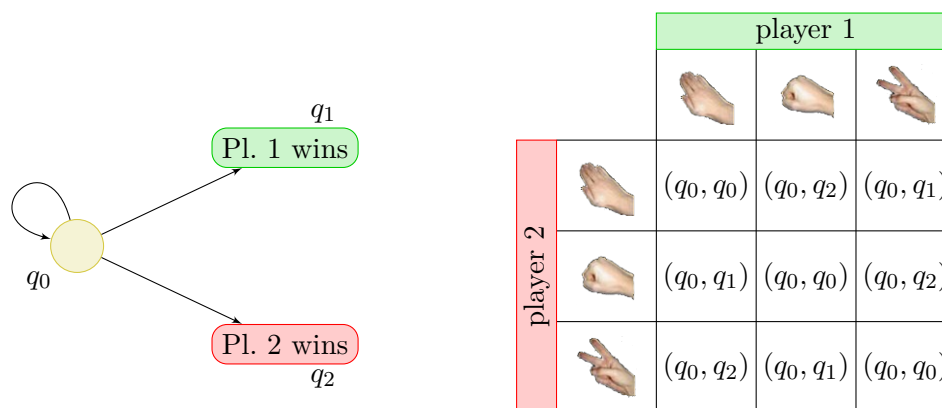


Figure 1.2: Example of the *paper-rock-scissors* automaton

particular timed games (which will be mentioned in the next section): in such games, the players may want (or have) to delay for some time in the current state, before deciding to move to some other state. The time at which the next transition is taken is then chosen concurrently.

Turn-based games are a special kind of concurrent games where in each state, all the players but possibly one have only one available move. Notice that, contrary to concurrent games whose transition table has size exponential in the number of agents, turn-based have size linear in the number of agents.

As sketched above, the evolution of a CGS is as follows: from a given state, each player simultaneously proposes one of its allowed moves. The next transition is then looked up in the transition table Tab according to this *move vector*.

An important object when dealing with games is strategies: a *strategy* for an agent A in a state s is a mapping from the set of finite runs of the CGS from s to a possible move from the last state of this run. In other terms, this is a labelling of the execution tree from s with moves for this player. At each node, the selected move restricts the set of possible successors, so that a strategy can also be seen as a subtree of the computation tree, containing those runs that are *compatible* with that strategy. We write $\text{Tree}_S(s, \sigma)$ for the subtree of $\text{Tree}_S(s)$ defined by a given strategy σ , and $\text{Strat}(s, A)$ for the set of strategies of A in s .

1.2.2 Verifying controllability and bisimilarity

Model checking has been extended to reason about multi-agent systems: temporal logics have been extended to express *controllability properties* such as: “agent A has a strategy to enforce a visit to a given state”. This new logic, called *full alternating-time temporal logic* (ATL*) [AHK97, AHK02], is an extension of CTL*. Its syntax is given by

$$\begin{aligned} \text{ATL}^* \ni \varphi_s &::= p \mid \neg \varphi_s \mid \varphi_s \vee \varphi_s \mid \langle\langle A \rangle\rangle \varphi_p \mid \llbracket A \rrbracket \varphi_p \\ \varphi_p &::= \varphi_s \mid \neg \varphi_p \mid \varphi_p \vee \varphi_p \mid \mathbf{X} \varphi_p \mid \varphi_p \mathbf{U} \varphi_p \end{aligned}$$

where p ranges over $\text{AP} \cup \{\text{true}\}$ and A over 2^{Agt} . ATL* formulas are interpreted along the computation tree of the CGS under study: the semantics is the same as for CTL*, with the following new rules:

$$\begin{aligned} \mathcal{T} \models \langle\langle A \rangle\rangle \varphi_p &\Leftrightarrow \exists \sigma \in \text{Strat}(\text{root}(\mathcal{T}), A). \forall w \in \text{br}(\text{Tree}_S(\text{root}(\mathcal{T}), \sigma)). w \models \varphi_p \\ \mathcal{T} \models \llbracket A \rrbracket \varphi_p &\Leftrightarrow \forall \sigma \in \text{Strat}(\text{root}(\mathcal{T}), A). \exists w \in \text{br}(\text{Tree}_S(\text{root}(\mathcal{T}), \sigma)). w \models \varphi_p \end{aligned}$$

ATL is the fragment of ATL* where temporal modalities are always associated with a strategy quantifier. Notice that ATL embeds CTL as the empty strategy quantifier $\langle\langle \emptyset \rangle\rangle$ corresponds to the universal path quantifier, while $\langle\langle \text{Agt} \rangle\rangle$ corresponds to the existential path quantifier (because our CGSs are deterministic, returning exactly one transition to a total move of Agt).

Model checking ATL is achieved in a way similar to CTL: we label states with the subformulas they satisfy. Computing the set of states satisfying $\langle\langle A \rangle\rangle \varphi \mathbf{U} \psi$ (for instance) is achieved in the same way as for CTL, except that the set of predecessors has to be replaced by the set of *controllable* predecessors. We define this set in two steps: first, given a state s , a coalition C and a move vector $m: C \rightarrow \text{Mov}$, we let

$$\text{Next}(s, C, m) = \{s' \in S \mid \exists \bar{m}: \text{Agt} \setminus C \rightarrow \text{Mov}. \text{Tab}(s, m \oplus \bar{m}) = (s, s')\}.$$

This is the set of states that can be reached when coalition C plays according to m . Then

$$\text{CPre}_C(X) = \{s \in S \mid \exists m: C \rightarrow \text{Mov}. \text{Next}(s, C, m) \subseteq X\}$$

(in other terms, it is the set of states from which players in coalition C can force the game to go in X in one step). Computing $\text{CPre}_C(X)$ is achieved by perusing the transition

table. We proved that it could be achieved in AC^0 , which is a subclass of PTIME (corresponding to unbounded fan-in boolean circuits with bounded depth). In the end, ATL model checking can be solved in PTIME.

ATL* can also be handled by a labelling algorithm. However, this requires computing the set of states satisfying $\langle\langle A \rangle\rangle \varphi$ for a formula φ in LTL. This can be achieved using tree automata. We will come back on this algorithm at Section 4.2.2).

Satisfiability of ATL has also been studied: the two important results to prove decidability of satisfiability are an encoding of ATL formulas into alternating tree automata on the one hand, and a bounded-branching model property of ATL, resulting in a finite-model property [GvD06]. This algorithm runs in 2-EXPTIME if the number of agents is not bounded, and has been improved to an EXPTIME algorithm in [WLWW06], using a type-elimination procedure. Finally, ATL* has been proved decidable in 2-EXPTIME in [Sch08] (using alternating tree automata).

To summarize:

Theorem ([AHK02, GvD06, WLWW06, Sch08]). • *ATL model checking is PTIME-complete, and satisfiability is EXPTIME-complete;*

- *ATL* model checking is 2-EXPTIME-complete, as well as satisfiability.*

Finally, let us mention that bisimulation has been extended to the game setting [AHKV98]: two states are *alternating bisimilar* when they carry the same labelling and any strategy of any coalition from one of the states can be mimicked from the other state (*mimicked* here meaning that the states reached by the mimicking strategy yields states that are alternating bisimilar to some state reached by the initial strategy).

As for CTL, alternating bisimilarity is related to ATL and ATL* in that two alternating-bisimilar states satisfy the same set of ATL and ATL* formulas. Finally, deciding alternating bisimilarity can be achieved in the same way as for standard bisimilarity, by a fixpoint computation.

Theorem ([AHKV98]). *Deciding alternating-bisimilarity is PTIME-complete.*

1.3 Modeling and verifying timed systems

Another important component of embedded systems is time (and more generally, quantitative notions for modelling the usage of resources). The (real-valued) quantitative study of these aspects started twenty years ago with the introduction of timed automata.

1.3.1 Timed automata

In order to model real-time systems, transition systems have been extended with real-valued clocks. These clocks will be used to modify the availability of transitions, via *timing constraints*. Given a finite set \mathcal{X} of real-valued variables (called *clocks*), timing constraints are built on the following grammar:

$$\text{TConstr}(\mathcal{X}) \ni g ::= x \sim c \mid g \wedge g$$

That a valuation $v: \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ satisfies a timing constraint is defined in the natural way. Two operations on clock valuations will be useful in the sequel: time shift correspond to adding a value t to all clocks: $(v + t)(c) = v(c) + t$ for all $c \in \mathcal{X}$; clock resets consist in setting some clocks to zero: $v[R \rightarrow 0]$ maps clocks in R to zero, and the other clocks to their values given by v .

A *timed automaton* [AD90, AD94]³ is a 4-tuple $\mathcal{T} = \langle \mathcal{S}, \mathcal{X}, \text{cond}, \text{inv} \rangle$ where \mathcal{S} is the underlying transition system, \mathcal{X} is a finite set of real-valued variables (called *clocks*), $\text{cond}: T \rightarrow 2^{\text{TConstr}(\mathcal{X}) \times 2^{\mathcal{X}}}$ labels each transition with one or several timing constraints, each being associated with a set of clocks to reset. Finally, $\text{inv}: \mathcal{S} \rightarrow \text{TConstr}(\mathcal{X})$ imposes timing constraints in states of the automaton.

Example. An example of a (two-clock) timed automaton is displayed on Figure 1.3. In this example, each transition carries only one pair in $\text{TConstr}(\mathcal{X}) \times 2^{\mathcal{X}}$, whose second component is written as “ $x := 0$ ” to indicate that the corresponding clock x is reset upon taking this transition.

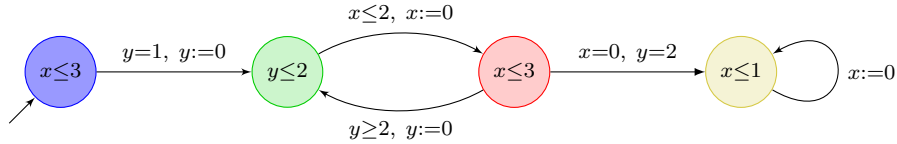


Figure 1.3: Example of a two-clock timed automaton

A configuration of a timed automaton \mathcal{A} is a pair (s, v) where s is a state of the automaton, v is a clock valuation, and $v \models \text{inv}(s)$. Since clock valuations take real values, timed automata usually have infinitely many configurations, and are in fact a finite representation of an infinite transition system $\mathcal{S}_{\mathcal{A}} = \langle S', T', \text{label}' \rangle$ where S' is the set of configurations, label' labels configurations according to its state and following label, and $T' \subseteq S' \times S'$ is the set of transitions, defined as follows: $((s, v), (s', v')) \in T'$ if, and only if, one of the following two conditions hold⁴:

- either it corresponds to time elapsing (*delay transition*), with no change in the state of the automaton: formally, this is expressed as

$$s = s' \text{ and } \exists t \in \mathbb{R}_{\geq 0}. v' = v + t$$

- or it correspond to the firing of a transition of the timed automaton (*action transition*):

$$\exists (g, r) \in \text{cond}(s, s'). v \models g \text{ and } v' = v[r \rightarrow 0].$$

The notion of a *run* of a timed automaton can be inherited from the above infinite-state transition system: a run of a timed automaton \mathcal{A} is a run in the corresponding

³State invariants were not present in the early definitions of timed automata, and were introduced in [HNSY94].

⁴Notice that invariant conditions are enforced by (s, v) and (s', v') being in S' .

transition system \mathcal{S}_A . In order to keep track of time elapsing, we assume that timed automata are equipped with a special clock t , which is reset along each transition and not used in the timing constraints of the automaton (hence t records the time elapsed in each state). The notion of *execution tree* follows.

While we consider real-time, the above notions of runs and trees are *discrete*: a run is a discrete sequence of configurations. It also makes sense to consider *continuous runs*, *i.e.*, functions mapping $\mathbb{R}_{\geq 0}$ to S (a.k.a. *signals*), hence keeping track of the visited state at any point in time. This gives rise in a very natural way to *dense trees*, which are the dense counterpart of classical discrete trees. In that setting, in order to preserve a way of representing dense runs with discrete information, it is generally assumed that signals have finite variability. This corresponds to the restriction to non-Zeno runs (*i.e.*, infinite runs along which time does not diverge, named after Zeno's paradox of Achilles and the Tortoise) in the discrete semantics.

In the sequel, we consider both semantics, referring to the integer-based semantics as the *timed-word semantics*, and to the signal-based semantics as the *signal semantics*. In order to have a uniform presentation (as much as possible), we define the following notations:

- in the timed-word semantics, given a run $\rho = (s_i, v_i)_i$ where for all i , s_i is a state of the timed automaton and v_i is a valuation of the clocks of the automaton, and given an index $k \in \mathbb{N}$, we let

$$\rho_{\text{AP}}(k) = \text{label}(s_k) \qquad \rho_{\tau}(k) = \sum_{j=1}^k v_j(t).$$

The suffix of ρ after position k is the run $\rho_{\geq k} = (s_{i+k}, v_{i+k})_i$.

- in the signal semantics, given a signal $\sigma: \mathbb{R}_{\geq 0} \rightarrow S$ and a positive real $k \in \mathbb{R}_{\geq 0}$,

$$\sigma_{\text{AP}}(k) = \text{label}(\sigma(k)) \qquad \sigma_{\tau}(k) = k.$$

The suffix of σ after k is the signal $\sigma_{\geq k}$ defined by $\sigma_{\geq k}(t) = \sigma(t + k)$.

1.3.2 Model checking timed properties and timed bisimilarity

Since timed automata represent infinite-state systems, already reachability checking is not obvious. An algorithm was proposed in [AD90], using as main tool the *region equivalence*, which will reveal the central tool for many problems on timed automata. The idea is as follows: the sets of runs from two *close* clock configurations are almost identical. More precisely, fix an integer vector $\mathbf{M} = (M_c)_{c \in \mathcal{X}}$. Two clock valuations v and v' are said to be *\mathbf{M} -region equivalent* (or *region equivalent* when \mathbf{M} is clear from the context) when the following holds:

- for all $c \in \mathcal{X}$, $v(c) > M_c$ iff $v'(c) > M_c$;
- for all $c \in \mathcal{X}$, if $v(c) \leq M_c$ and $v'(c) \leq M_c$, then the integral parts $\lfloor v(c) \rfloor$ and $\lfloor v'(c) \rfloor$ are equal, and the fractional part $\langle v(c) \rangle$ is zero iff the fractional part $\langle v'(c) \rangle$ is also zero;

- for any two clocks c and c' with $v(c) \leq M_c$, $v'(c) \leq M_c$, $v(c') \leq M_{c'}$ and $v'(c') \leq M_{c'}$, it holds $\langle v(c) \rangle < \langle v'(c) \rangle$ iff $\langle v'(c) \rangle < \langle v'(c') \rangle$.

We write $[v]_{\mathbf{M}}$ for the equivalence class of valuation v (but we will often omit to mention \mathbf{M} when no ambiguity arises).

Example. Figure 1.4 displays the set of regions for two clocks x and y with maximal-constant vector $(2, 2)$. There are 44 regions in this case.

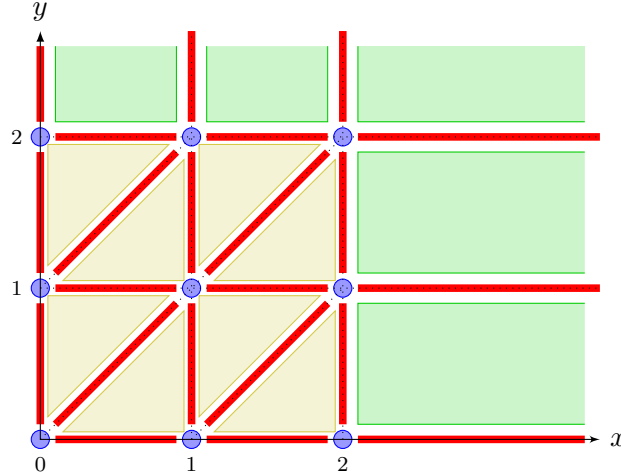


Figure 1.4: A representation of the set of regions for two clocks

Two \mathbf{M} -region-equivalent valuations v and v' satisfy the following three conditions:

- for all $t \in \mathbb{R}_{\geq 0}$, there exists $t' \in \mathbb{R}_{\geq 0}$ s.t. $v + t$ and $v' + t'$ are region equivalent;
- for all clock constraint g in which each clock c is compared to integers less than or equal to M_c , it holds $v \models g$ if, and only if, $v' \models g$;
- for all $r \subseteq \mathcal{X}$, $v[r \rightarrow 0]$ and $v'[r \rightarrow 0]$ are \mathbf{M} -region-equivalent.

As a consequence, letting \mathbf{M} consist of the maximal integer constants each clock is compared with, for any state s and any two \mathbf{M} -region-equivalent valuations v and v' , the configurations (s, v) and (s, v') are bisimilar. The transition system obtained by quotienting the semantics $\mathcal{S}_{\mathcal{A}}$ with the region equivalence is then also bisimilar to $\mathcal{S}_{\mathcal{A}}$.

Moreover, this new transition system, called the *region automaton* and denote with $\mathcal{R}_{\mathcal{A}}$, is finite-state: indeed, the number of equivalence classes for the region equivalence can be shown to be bounded by $|\mathcal{X}|! \cdot 2^{|\mathcal{X}|} \cdot \prod_{c \in \mathcal{X}} (2 \cdot M_c + 2)$ [AD94].

Example. For our example of Figure 1.3, the corresponding region automaton is represented at Figure 1.5. Only the reachable part has been represented (which shows in particular that the rightmost state is not reachable). Notice that, for the sake of readability, only delay transitions from one region to the immediate successor are drawn.

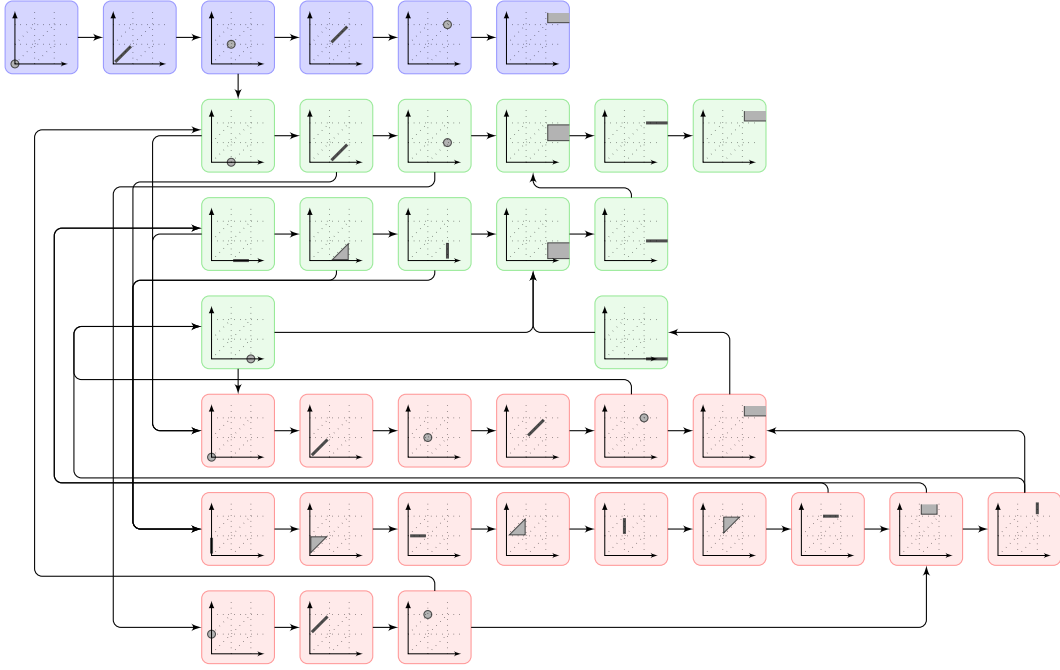


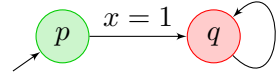
Figure 1.5: The region automaton associated with the timed automaton of Figure 1.3.

The semantics of CTL and LTL depend on the semantics assumed for timed automata. Actually, besides the timed-word vs. signal semantics, many slight variants of the semantics have been proposed, depending for instance on the strictness of the “Until” modality, or its “matchingness” (whether $\varphi \wedge \psi$ or only ψ must hold to witness the eventuality in formula $\varphi \mathbf{U} \psi$) [FR07]. Here we assume strict, non-matching semantics of “until”:

$$w \models \varphi \mathbf{U} \psi \quad \Leftrightarrow \quad \exists j > 0. w_{\geq j} \models \psi \text{ and } \forall 0 < i < j. w_{\geq i} \models \varphi$$

(irrespective of whether w is a discrete run or a signal).

The denseness of the time line still plays an important role in the semantics: consider the CTL formula $\mathbf{E}((\neg p) \mathbf{U} \mathbf{E} \mathbf{G} q)$, and the timed automaton \mathcal{A} beside. One would expect this formula to not hold in the initial configuration of this automaton, since p will hold for the first time unit, and $\mathbf{E} \mathbf{G} q$ will not hold before that. However, the transition system $\mathcal{S}_{\mathcal{A}}$ has a transition from $(p, x = 0)$ to $(p, x = 1)$ and a transition from $(p, x = 1)$ to $(q, x = 1)$, witnessing the fact that this formula does hold in \mathcal{A} under the timed-word semantics. Under the signal semantics, the subformula $\neg p$ will be checked at all intermediary time points between $x = 0$ and $x = 1$, so that the formula will fail to hold under this semantics.



From Theorem 1.1.2, it follows that CTL formulas to be checked on (the semantics of) a timed automaton can be checked on its region automaton, which provides us with an algorithm for CTL model checking under the timed-word semantics. Non-Zenoness

condition can be enforced by adding an extra `tick` clock and checking that it ticks infinitely many times along infinite runs. Under the signal semantics, a similar labelling algorithm can be achieved, with the small difference that it will check all intermediate regions visited during a delay transition [ACD93].

Concerning LTL, we also handle both semantics differently:

- under the timed-word semantics: this semantics corresponds to checking the LTL property on the infinite-state transition system $\mathcal{S}_{\mathcal{A}}$. Since bisimulation subsumes trace equivalence, LTL properties can also be checked on the region automaton;
- under the signal semantics: LTL model checking can be achieved by considering the timed automaton obtained as the product of the timed automaton \mathcal{A} under checking with the Büchi automaton for the LTL formula. Checking non-emptiness of the resulting timed automaton with a Büchi condition can be achieved in polynomial space through the region abstraction.

Hence:

Theorem ([ACD93, AD94]). *CTL and LTL model checking on timed automata are PSPACE-complete.*

Notice that, for a fixed model, CTL model checking is solvable in PTIME, while LTL model checking is PSPACE-complete. The overall PSPACE complexity comes from the fact that the region automaton has size exponential in the size of the timed automaton it represents.

Obviously, quantitative properties would be even nicer. Temporal logics have been extended with quantitative timing constraints in two different ways:

- with constrained modalities [Koy87]: for instance, formula $\varphi \mathbf{U}_{[3,5]} \psi$ states that $\varphi \mathbf{U} \psi$ holds, with the additional constraint that the witnessing position for ψ must occur within 3 to 5 time units. Formally, given a (discrete or continuous) run w , that $w \models \varphi \mathbf{U}_I \psi$ is defined as

$$\exists j > 0. w_{\geq j} \models \psi \text{ and } \forall 0 < i < j. w_{\geq i} \models \varphi \text{ and } w_{\tau}(i) \in I.$$

- with formula clocks [AH89, ACD93]: in that setting, formulas involve extra variables (*not* clocks from the timed automaton), which can be reset and later compared to an integer. For instance, the above property $\varphi \mathbf{U}_{[3,5]} \psi$ would be written here $x.(\varphi \mathbf{U} (\psi \wedge x \in [3, 5]))$, where x . corresponds to resetting x . Formally, this requires keeping track of the values of the formula clocks when evaluating formulas. This information is stored in a valuation ζ in the semantics:

$$\begin{aligned} w \models_{\zeta} x.\varphi_p &\Leftrightarrow w \models_{\zeta[x \rightarrow 0]} \varphi_p \\ w \models_{\zeta} x \in I &\Leftrightarrow \zeta(x) \in I \\ w \models_{\zeta} \varphi \mathbf{U} \psi &\Leftrightarrow \exists j > 0. w_{\geq j} \models_{\zeta + w_{\tau}(j)} \psi \text{ and } \forall 0 < i < j. w_{\geq i} \models_{\zeta + w_{\tau}(i)} \varphi \end{aligned}$$

Both extensions can be applied to branching-time and linear-time temporal logics. We write TCTL and TCTL_c for the extensions of CTL with constrained modalities and formula clocks, respectively. Notice that, following the sample formula above, any TCTL formula can be written as a TCTL_c formula. The extension of LTL with constrained modalities is called MTL (*metric temporal logic*), while the extension with formula clocks is called TPTL (*timed propositional temporal logic*).

Model checking TCTL_c (and hence TCTL) is decidable: again, we rely on an algorithm labelling the states of the (extended) region automaton with state subformulas. The region automaton has to be augmented with the formula clocks and their corresponding maximal constants. This provides us with a way of taking care of the values of the formula clocks while checking subformulas. In the end:

Theorem ([ACD93]). *TCTL and TCTL_c model checking on timed automata are PSPACE-complete.*

Notice that this time, the problem is also PSPACE-complete over fixed timed automata.

In practice, the exponential number of regions is prohibitive for applying this method. This can be overcome by using *zones*, which are sets of valuations defined as conjunctions as clock constraints (including *diagonal constraints*, comparing the difference of two clocks with integers). Zones can be manipulated efficiently through *Difference Bound Matrices* (DBMs), and provides us with a way of handling several regions at the same time, thus making the algorithms more efficient in practice (despite the fact that the number of zones is larger than the number of regions). This has been implemented in several tools (UppAal, Kronos, ...) [BDL05, Yov97], and successfully applied on many industrial case studies [HSL97, BGK⁺96].

Regarding linear-time, the situation is less appealing: model checking is undecidable for both MTL and TPTL. This can be proved by encoding the behaviour of a Turing machine [AH92b]: each configuration of the Turing machine is encoded on a one-time-unit segment, with the content of the machine's tape encoded as a sequence of events along this segment. Considering real-time allows us to encode unboundedly many cells of the Turing machine. Transitions are then encoded by preserving most of the configuration one time unit later, updating only the segment around the machine's tape head.

As remarked by Joël Ouaknine and James B. Worrell in [OW05], this reduction actually only works under the signal semantics: under the timed-word semantics, it is not possible to prevent *insertion errors*, *i.e.*, new cells inserted in the encoding of a configuration while they were not present in the encoding of the previous configuration. Joël Ouaknine and James B. Worrell proved that undecidability still holds under the timed-word semantics, using channel automata with insertion errors [OW06a]. Notice that the same authors also proved *decidability* of MTL under the timed-word semantics when considering finite runs of timed automata (assuming that they have accepting states), which is not the setting we have adopted here. We will come back on these results in Section 2.2.

To sum up, we have:

Theorem ([AH92b, OW05, OW06a]). *MTL and TPTL model-checking problems are undecidable under both the timed-word and signal semantics.*

Regarding satisfiability, it can be observed that the above undecidability proof for MTL straightforwardly adapts to satisfiability. For TCTL also, satisfiability can be proved undecidable, using a very similar encoding [ACD93].

Theorem ([ACD93, AH92b]). *Satisfiability is undecidable for TCTL, MTL and TPTL.*

The above undecidability proofs make heavy use of punctual constraints for replicating (except for small changes around the tape head) successive encodings of the configurations of the Turing machine under study. With the hope of recovering decidability, a fragment of MTL banning punctual constraints has been defined: MITL (for *Metric Interval Temporal Logic*) is the fragment of MTL in which timing constraints must be non-punctual intervals. MITL has been proved to have a decidable model-checking and satisfiability problems, by a transformation of MITL formulas into timed automata: in the case of MTL, an automaton for encoding $\mathbf{G}(p \Rightarrow \mathbf{F}_{=1} q)$ would use unboundedly many clocks, which would be reset each time a p -event is triggered. MITL modalities have *bounded variability*: formula $\mathbf{G}(p \Rightarrow \mathbf{F}_{[1,2]} q)$ is then checked by non-deterministically guessing the dates at which $\mathbf{F}_{[1,2]} q$ changes its status. For each one-time-unit interval, this requires two clocks that will be reset non-deterministically and used to check the presence of q -events witnessing the guess. The whole translation yields an exponential-size timed automaton accepting exactly the runs that satisfy the MITL formula under study.

Theorem ([AFH96]). *MITL model checking and satisfiability are EXPSpace-complete under both the timed-word and signal semantics.*

Several alternative proofs of this (or closely related) result using different techniques have been proposed in the literature [RS99, HR05, MNP06].

Let us mention that also for TCTL, banning equality constraints can be beneficial, as it makes finite satisfiability (*i.e.*, the existence of a satisfying automaton, as opposed to the existence of a satisfying infinite tree) decidable:

Theorem ([LN00, LN01]). *TCTL_{<,>} finite satisfiability is decidable in exponential time under both the timed-word and signal semantics.*

We conclude this section with trace-equivalence and bisimulation checking. In the timed setting, both notions comes with two flavours, with or without quantitative timings.

Regarding trace equivalence, since universality of timed automata is undecidable [AH94], timed-trace equivalence is undecidable. Untimed-trace equivalence is proved decidable in [ACH94], by observing that it can be checked on the region automaton.

Untimed bisimilarity is just classical bisimilarity (as defined in Section 1.1.2) on the infinite-state transition system representing the semantics of the timed automata under study. Untimed bisimilarity can be decided by a fixpoint computation similar to the classical bisimilarity-checking algorithm: it can be seen that all the sets of states computed by this algorithm are unions of regions, hence the termination of the algorithm in exponential time [ACH94, LY94]. Notice that obviously, untimed bisimilarity preserves CTL formulas, but does not preserve TCTL ones.

Timed bisimilarity extends bisimilarity with the requirement that the delays must match. Deciding timed bisimilarity is proved decidable in exponential time in [Čer93]. An alternative algorithm can be obtained by reducing the problem to the existence of a winning strategy in a two-player turn-based timed game (see Section 1.3.4), in quite the same way as explained at the end of Section 1.1.3.

Theorem ([Čer93, ACH94, LY94]). *Timed-trace-equivalence checking is undecidable. Untimed-trace-equivalence checking is decidable in EXPSpace. Timed- and untimed-bisimilarity checking are EXPTIME-complete.*

1.3.3 Weighted timed automata

Since their introduction in the last 1980's, timed automata have received (and continue to receive) much attention from the formal-verification community: most of the important decision problems now have their (theoretically) optimal algorithm, and on the practical side, efficient tools have been developed and are being used successfully on numerous industrial case studies.

One important feature that is missing in timed automata is the ability to *measure* accumulated delays and other quantities. Hybrid automata have been introduced in the early 1990's as an extension of timed automata with richer dynamics: (hybrid) variables are now allowed to have different slopes, depending on the configuration of the automaton. This may result in variables obeying differential inclusions, which most often leads to undecidability.

Actually, it has been shown that even the simple extension of timed automata with one single stopwatch variable has undecidable reachability [HKPV98]. As is usual in this setting, this is proved by encoding the halting problem of two-counter machines: the hybrid automaton will have three clocks—one serving as a `tick` clock, and the other two are used to store the values of the counters with the correspondence $x_i = 2^{-c_i}$ when the `tick` clock equals zero—and one stopwatch variable. It then suffices to build modules for incrementing and decrementing the counters, *i.e.*, doubling and halving the value of one of the encoding clocks (without losing the information stored in the other clock). The module for doubling the clock value (*i.e.* decrementing a counter) is depicted at Figure 1.6 (where we omitted to indicate self-loops on each state that allow x_2 to be reset when it reaches value 1. Since traversing this module takes 3 time units, we exit the module with x_2 unchanged, and x_1 is easily computed to double its value.

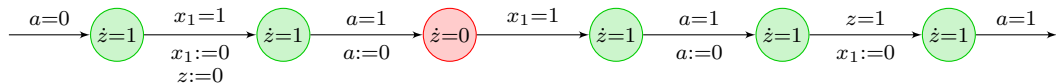


Figure 1.6: Stopwatch-automaton for doubling the value of a clock

This negative result leaves very little room for decidable classes of hybrid automata. Actually, two such classes have been identified:

- *initialized* rectangular automata [HKPV98], in which hybrid variables take their derivatives in different intervals (depending on states) and, more importantly,

are reset (to some value in a given interval) between two states with different dynamics. Initialized stopwatch automata can easily be transformed into timed automata by noticing that stopwatches are set to a new value when stopped and started, which can be simulated by a clock and some extra information stored in states. Initialized rectangular automata can in turn be transformed into initialized stopwatch automata by duplicating the number of variables (to have a lower- and an upper-bound) and stretching them to have slope 1 (or 0).

- two-dimensional polygonal differential inclusion systems, where the plane (representing the state space of the system) is partitioned into finitely many polyhedra in which the evolution of the trajectory is constrained between two angles. Reachability in these models are proved decidable in [ASY07], while it is undecidable with three hybrid variables [AMP95].

In 2001, Alur *et al.* and Larsen *et al.* independently introduced an extension of timed automata with hybrid variables for which they proved optimal reachability is decidable [ALP01, BFH⁺01]. The important difference between these and plain hybrid automata is that the dynamics in weighted automata with hybrid variables only depends on clocks: hybrid variables are *observers*: they can measure the elapse of time or other quantities in different states, but they cannot be used to constrain the availability of transitions. In other terms, *weighted timed automata* (as we will call them from now on) are simply timed automata whose states and transitions carry values indicating the “price to pay” to delay in this location or take that transition.

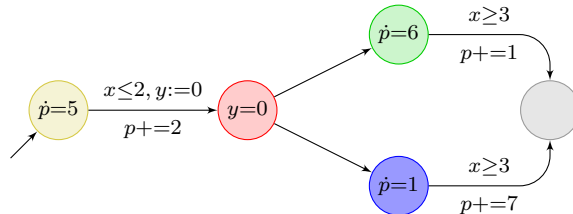


Figure 1.7: Example of a weighted timed automaton

Example. Figure 1.7 displays an example of a weighted timed automaton, with one single⁵ hybrid observer p . The possible runs of such an automaton only depend on the clock valuations, and each run is decorated with the value of the observer variable. For instance, delaying for 1.2 time units in the initial location involves an increase of 6 units in the value of the observer variable. It can be checked that it is possible to reach the final state with $p = 12$ (assuming that it is initially zero), by spending three time units in the bottom state (with $\dot{p} = 1$).

Besides reachability, which can be handled by just considering the underlying timed automaton, optimal reachability is the most natural problem for those models. The problem has been proved to be no harder than plain reachability in classical timed automata.

⁵Apart from very recent works, weighted timed automata have been studied in the setting of only one hybrid observer variable. This is the setting we will assume all along this document.

We briefly describe the *corner-point-abstraction* approach [BBL08] for the case where the value of the observer is non-decreasing: in that case, assuming that a finite path is fixed between the initial and the target state, the delays spent in each state along this path have to satisfy affine constraints derived from the guards of the underlying timed automaton. The set of possible delays form a bounded k -dimensional zone, where k is the length of the considered path. Now, the total weight along this path is a linear function of these delays. It can be proved, inductively on the number of variables, that infima of such functions lie on the border of the zone, at points with integer coordinates. In other terms, optimal runs have their transitions taken at (or close to, in case of guards involving strict inequalities) integer dates. Yet another way of seeing this is as follows: if we start spending some time in a state, then there is no reason for escaping as long as no new transition is made available.

This observation has led to the definition of the *corner-point automaton* [BBL08], which refines the region automaton with the additional indication, in each region, of one of its corners. This finite-state abstraction is sufficient for finding optimal runs in weighted timed automata, and provides us with a PSPACE algorithm for this problem.

Theorem ([ALP01, BFH⁺01, BBL08]). *The optimal reachability problem is PSPACE-complete.*

1.3.4 Timed games

Timed games are another extension of timed automata, with a way to model interactions of different entities on the evolution of the underlying timed automaton.

Timed games have mostly been studied in the setting of two players: the pioneering work in this setting is [MPS95], where timed games are defined as timed automata in which the set of transitions is partitioned into transitions belonging to Player 1 (said to be *controllable*) and transitions belonging to Player 2 (uncontrollable). A strategy for Player 1 maps each finite run ρ to a controllable transition (available from the last configuration of ρ), or to *wait* (meaning that this player wants to let time elapse). Such a strategy generates a subset of the runs of the timed game, which must satisfy Player 1's objective in order to be declared winning.

The existence of a (memoryless) winning strategy in two-player timed games with simple objectives (reachability, safety, ...) has been proved decidable in exponential time (by a fixpoint computation on the finite set of regions):

Theorem ([MPS95]). *The existence of a winning strategy in a timed game automaton with reachability (resp. safety) objective is decidable.*

Other algorithms have been proposed to solve the problem *on-the-fly*, without visiting the whole state space [TA99, CDF⁺05], and have been implemented in the tool UppAal TIGA.

Several other semantics for timed games have been proposed [AMPS98, dAFH⁺03]. In [dAFH⁺03], contrary to the previous setting, a move for a player is a pair made of a delay (possibly zero) and an action to be taken after this delay. Once all the players have proposed such a move, the player with the smaller delay is elected to apply her

move, and the game continues from the resulting configuration. This definition is, in some sense, more *symmetric*, since it considers strategies of both players. Actually, this semantics of timed games can be expressed as an infinite-state two-player CGS, in quite the same way as the semantics of timed automata can be defined in terms of an infinite-state transition system. More importantly, this semantics has been extended with a way of ruling out Zeno strategies: such strategies are detected by giving a blame at each round to the player applying its transition, so that non-Zeno strategies are those that receive only finitely many blames along any time-converging outcome. In order to be winning, a strategy must enforce the objective and be non-Zeno. Using fixpoint computation on the region abstraction, the following result can be obtained:

Theorem ([dAFH⁺03]). *The existence of non-Zeno winning strategies in timed games with parity objectives (on states) is EXPTIME-complete.*

2

Verification of timed systems

Contents

2.1	Expressiveness of temporal logics	32
2.1.1	Expressiveness of MITL	32
2.1.2	Expressiveness of TPTL and MTL	33
2.2	Real-time model-checking with punctuality	37
2.2.1	Decidable fragments of MTL	37
2.2.2	coFlatMTL in the timed-word semantics	41
2.2.3	coFlatMTL in the signal semantics	46
2.3	Robust model-checking	48
2.3.1	Implementability of timed automata	48
2.3.2	Robust verification of safety properties	50
2.3.3	Robust verification of LTL and coFlatMTL	53
2.4	Conclusions and future works	55

Most of the main results about real-time model-checking have been established in the early years after the introduction of timed automata: the branching-time logic TCTL is decidable in PSPACE [ACD93], with a region-based labelling algorithm, while linear-time logics are in general undecidable, with the notable exception of MITL, in which punctual constraints are banned [AFH96]. The problem with punctual constraints is that they provide a way to repeat a pattern every time unit, by saying that “*any a-event has a corresponding a-event one time unit later*”. Using this trick, it is possible to encode a computation of a Turing machine as a timed word: the i -th configuration of the machine is encoded by the part of the timed word between dates i and $i + 1$. That a configuration

is the successor of the previous one by some transition of the Turing machine can be expressed by saying that the events between dates i and $i + 1$ repeat one time unit later, except around the position representing the tape head. We refer the reader to [AH92b] for further details.

However, several questions were left open, or not completely sorted out. This chapter presents my contributions in three directions: on expressiveness first, where we solve a fifteen-year-old conjecture on the relative expressiveness of TPTL and MTL; then on model-checking algorithms, where we defined a decidable timed temporal logic *with* punctual constraints; finally, we studied an alternative semantics of timed automata, taking into account the imprecision and finite frequency of the platform on which it is implemented.

2.1 Expressiveness of temporal logics

Expressiveness results are generally intricate and difficult to establish (see Table 1 for some explanations on expressiveness and classical techniques to tackle expressiveness problems). Timed temporal logics are not an exception: several quantitative extensions of LTL have been defined in the early 1990’s, but only few expressiveness results (apart from the obvious ones) had been answered. The notable exceptions concern the logic MITL. We begin with a short review of these existing results, before presenting our result on the relative expressiveness of MTL and TPTL.

2.1.1 Expressiveness of MITL

A first exception concerns the relative expressiveness of MITL and MTL. The difference in the decidability of their model-checking problems implies that one cannot *effectively* translate an MTL formula into an equivalent MITL one. This does not imply, *stricto sensu*, that MTL is more expressive than MITL. However, this stronger result can be derived from the following two facts: on the one hand, it is known from [AD94] that there does not exist a timed automaton that would accept exactly those timed words in which the distance between any two events never equals 1 time unit¹. That no two events are exactly 1 time unit apart can be expressed in MTL with $\neg \mathbf{F}(p \wedge \mathbf{F}_{=1} p)$. On the other hand, from [AFH96], for any MITL formula φ , we can compute a timed automaton \mathcal{B}_φ that accepts exactly the models of φ under the signal semantics. The same holds in the timed-word semantics from [AH92a]. It follows that MTL is strictly more expressive than MITL.

A second expressiveness result relates MITL and its extension with past-time modalities: contrary to what happens with LTL [Kam68, GPSS80], past-time modalities (with non-singular interval constraints) do add expressiveness to MITL. Using reversal-bounded two-way timed automata, Rajeev Alur and Thomas Henzinger proved the following result: for all $k \in \mathbb{N}$, MITL+Past $_{k+2}$ is strictly more expressive than MITL+Past $_k$ under the timed-word semantics (where MITL+Past $_k$ is the fragment of MITL+Past with at most k nested alternations of “until” and “since” modalities) [AH92a].

¹In [AD94], this is only stated for the timed-word semantics, but it clearly extends to the signal semantics.

Finally, several expressiveness results for MITL are proved in [Ras99], drawing tight links between this logic and ECTL, an extension of LTL with quantitative modalities to refer to the next and previous occurrence of an event. Regarding MITL, we mention the following result: MITL and MITL_~ are equally expressive in the signal semantics, but MITL is strictly more expressive than MITL_~ in the timed-word semantics (where MITL_~ is the fragment of MITL in which modalities are decorated with inequality constraints instead of (non-punctual) interval constraints).

2.1.2 Expressiveness of TPTL and MTL

Regarding the relative expressiveness of MTL and TPTL, the obvious inclusion was folklore: any MTL formula can be translated into TPTL (with one formula clock), since we have the following equivalence:

$$\varphi_1 \mathbf{U}_{\langle a,b \rangle} \varphi_2 \equiv x.(\varphi_1 \mathbf{U}(\varphi_2 \wedge x \in \langle a, b \rangle)) \quad (2.1)$$

(which is valid both in the signal- and timed-word semantics). It was conjectured by Rajeev Alur and Thomas Henzinger that the converse inclusion would not hold, with the formula

$$x.\mathbf{F}(p \wedge \mathbf{F}(q \wedge x \leq 2)) \quad (2.2)$$

as a candidate not being expressible in MTL [AH92b, AH93, Hen98]. This formula states that two events p and q will occur, in that order, in the next two time units. The obvious tentative equivalent formulas in MTL fail: $\mathbf{F}_{[0,2]} p \wedge \mathbf{F}_{[0,2]} q$ does not enforce that p occurs before q ; $\mathbf{F}_{[0,2]}(p \wedge \mathbf{F}_{[0,2]} q)$ allows the witnessing q event to occur later than required.

It turns out that Formula (2.2) can be expressed in MTL, assuming the signal semantics, as follows [BCM05]:

$$(\mathbf{F}_{[0,1]} p \wedge \mathbf{F}_{[1,2]} q) \quad \vee \quad \mathbf{F}_{[0,1]}(p \wedge \mathbf{F}_{[0,1]} q) \quad \vee \quad \mathbf{F}_{[0,1]}(\mathbf{F}_{=1} q \wedge \mathbf{F}_{[0,1]} p). \quad (2.3)$$

Why this formula is equivalent to Formula (2.2) is schematically depicted on Figure 2.1, where we distinguish between three eventualities:

- if p occurs before time 1, then two cases may arise:
 - either q occurs in $[1, 2]$, which is captured by the first conjunct in Formula (2.3);
 - or it occurs before time 1, hence less than 1 time unit after p , which is expressed by the second disjunct²;
- otherwise, p occurs after time 1: in that case, the delay between the p and the q event is less than 1, so that the time instant that lies one time unit before q lies before p . At that time, p will occur in less than one time unit, and q in exactly one time unit. This is captured by the third disjunct of Formula (2.3).

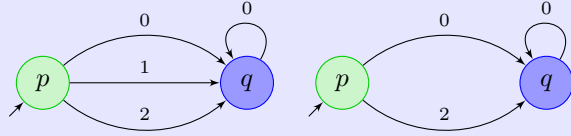
It must be noticed that this translation of formula (2.2) crucially uses the signal semantics, by referring to the time instant situated one time unit before the q -event. It turns out that formula (2.2) cannot be expressed in the timed-word semantics:

²Actually the second disjunct captures more than just this, as it overlaps with the first conjunct.

A temporal logic \mathcal{L} is said to be *at least as expressive* as another one \mathcal{L}' if for any formula φ of \mathcal{L} , there is a formula φ' of \mathcal{L}' which is equivalent to φ (i.e., it has the same truth value in any model). If also \mathcal{L}' is at least as expressive as \mathcal{L} , then both logics are said to be *equally expressive*. Otherwise, \mathcal{L} is *strictly more expressive* than \mathcal{L}' . Notice that, stricto sensu, this depends on the underlying class of models. For instance, while MTL is strictly more expressive than MITL over timed automata under dense-time semantics, both logics are equally expressive under discrete-time semantics (with “strict until”): for example, formula $\mathbf{F}_{=1} \varphi$ is expressed as $\mathbf{F}_{\leq 0} (\perp \mathbf{U}_{\leq 1} \varphi \wedge \perp \mathbf{U}_{\geq 1} \varphi)$. Notice that the size of the resulting formula may grow exponentially, as we duplicate φ . Similarly, TPTL and MTL have been proved equally expressive over discrete time [AH92b].

Proving that two logics \mathcal{L} and \mathcal{L}' are equally expressive is usually achieved by providing translations in both directions (it is often the case that one direction is a syntactic inclusion). Proving that a logic \mathcal{L} is strictly more expressive than \mathcal{L}' often amounts to exhibiting a witness formula φ from \mathcal{L} and showing that no formula from \mathcal{L}' is equivalent to φ . The easiest way of proving this is by exhibiting two models \mathcal{M} and \mathcal{M}' such that $\mathcal{M} \models \varphi$ and $\mathcal{M}' \not\models \varphi$, and showing that, for all $\varphi' \in \mathcal{L}'$, $\mathcal{M} \models \varphi'$ if, and only if, $\mathcal{M}' \models \varphi'$.

Consider for instance the logic TCTL_{\sim} , where timing constraints are comparisons to integers (intervals not allowed). We can prove that $\mathbf{EF}_{=1} q$ cannot be expressed in TCTL_{\sim} without equality over discrete-time automata, as both models on the right cannot be separated by this logic (while obviously only the first one satisfies $\mathbf{EF}_{=1} q$).



This technique is very often not powerful enough: in many cases, two different models can be distinguished by the logic, for instance by nesting sufficiently many “next time” modalities to point to a position where both models differ (this characterizes the relative *distinguishing power* of temporal logics). For example, any two different words can be separated this way. In order to prove expressiveness results in this case, one idea is to consider *families of models* $(\mathcal{M}_i)_i$ and $(\mathcal{M}'_i)_i$, such that for all i , $\mathcal{M}_i \models \varphi$ and $\mathcal{M}'_i \not\models \varphi$, and \mathcal{M}_i and \mathcal{M}'_i cannot be distinguished by formulas of \mathcal{L}' of size less than i . We use this technique in Section 4.1.1 for ATL.

In the case of timed temporal logics over dense time, this may again not be sufficient: two timed words having respectively n and $n + 1$ evenly disposed p -events during the first time unit can be separated with formula $\mathbf{F}_{=\frac{1}{2}} p$. We want to allow rationals in timing constraints, since the expressive power of a logic should not depend on the time scale, when considering dense time. This requires doubly-indexed families of models $(\mathcal{M}_{i,j})_{i,j}$ and $(\mathcal{M}'_{i,j})_{i,j}$, one of which satisfies φ , and such that $\mathcal{M}_{i,j}$ and $\mathcal{M}'_{i,j}$ cannot be separated by a formula of size less than i and with *granularity* less than j (where the granularity is the least common denominator of the rational constants in the formula). Our expressiveness results in the timed setting rely on this technique (but in the proof of Proposition 1, the index for the size of the formula is not used).

Table 1: Expressiveness proof techniques

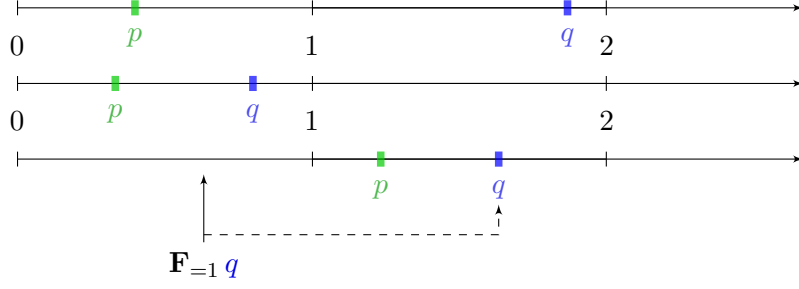


Figure 2.1: Expressing $x.\mathbf{F}(p \wedge \mathbf{F}(q \wedge x \leq 2))$ in MTL

Proposition 1. *The TPTL formula $x.\mathbf{F}(p \wedge \mathbf{F}(q \wedge x \leq 2))$ has no equivalent in MTL under the timed-word semantics.*

The proof of this result is based on two infinite families of models, depicted on Figure 2.2. While it is clear that Formula (2.2) holds along \mathcal{A}_n for any $n \in \mathbb{N}_{>0}$, but fails to hold along any \mathcal{B}_n . The proof consists in proving that MTL cannot distinguish between these two families. Actually, we aim at the stronger result that even MTL formulas with rational constants (with *granularity* $1/n$) cannot distinguish \mathcal{A}_n from \mathcal{B}_n . This proof is by induction on the structure of MTL formulas [BCM10].

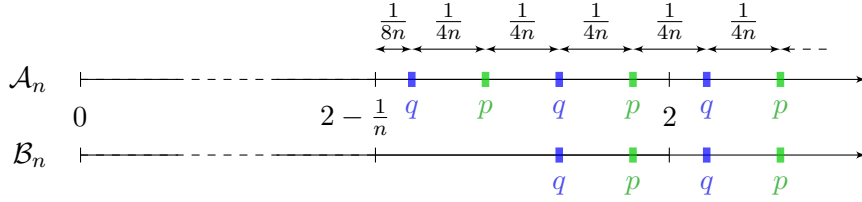


Figure 2.2: Two timed words that $x.\mathbf{F}(p \wedge \mathbf{F}(q \wedge x \leq 2))$ can distinguish

Since the families $(\mathcal{A}_n)_{n \in \mathbb{N}_{>0}}$ and $(\mathcal{B}_n)_{n \in \mathbb{N}_{>0}}$ can be distinguished by the MITL+Past formula $\mathbf{F}_{\leq 2}(q \wedge \mathbf{F}^{-1}p)$, this proof also provides the following results:

Corollary 2. *The MITL+Past formula $\mathbf{F}_{\leq 2}(q \wedge \mathbf{F}^{-1}p)$ has no equivalent in MTL under the timed-word semantics.*

We now turn to the signal semantics, and propose another formula for witnessing that indeed, TPTL is also strictly more expressive than MTL under that semantics.

Proposition 3. *The TPTL formula*

$$x.\mathbf{F}(p \wedge x \leq 1 \wedge \mathbf{G}(x \leq 1 \Rightarrow \neg q)) \quad (2.4)$$

has no equivalent in MTL under the signal semantics.

This formula states that the last non-empty event before time 1 is a p (assuming a two-letter alphabet). This proof is similar to the previous one, in that we define two families of models that we prove MTL cannot distinguish. However, both families are indexed by two integers here, as we have to take both nesting depth and granularity into account. Those families are depicted on Figure 2.3.

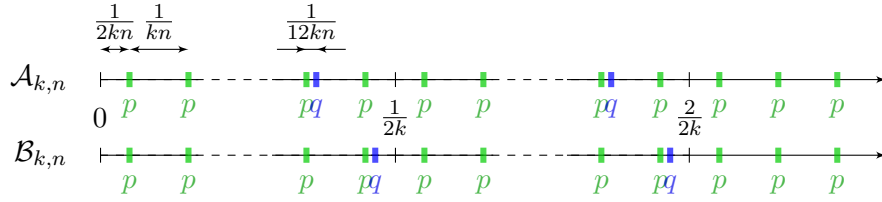


Figure 2.3: Two timed words that $x.\mathbf{F}(p \wedge x \leq 1 \wedge \mathbf{G}(x \leq 1 \Rightarrow \neg q))$ can distinguish

These models contain a few q -events ($2k$ such events per time unit) which are flooded by p -events (n occurrences of p for one occurrence of q). They can be proved indistinguishable by $\text{MTL}_{\sim}^{n,k}$, which is the fragment of MTL containing formulas of temporal height at most n (*i.e.*, having at most n nested temporal modalities) and only using timing constraints of the form $\sim c$ where $\sim \in \{<, =, >\}$ and $c \in \mathbb{Q}$ with $k' \cdot c \in \mathbb{N}$ for some $k' \leq k$. This fragment is easily shown to be at least as expressive as $\text{MTL}^{n/2,k}$, where interval are allowed in timing constraints.

To conclude the proof, it suffices to observe that Formula (2.4) holds true along any $\mathcal{A}_{k,n}$, but fails to hold along any $\mathcal{B}_{k,n}$.

Finally, as for the timed-word semantics, notice that we can separate both families of models with $\text{MTL}+\text{Past}$ and $\text{MITL}+\text{Past}$:

Corollary 4. *The MITL+Past formula*

$$\mathbf{F}_{\geq 1}(\neg p \wedge (\neg q \mathbf{S} p) \wedge \mathbf{F}_{\geq -1}^{-1}(\mathbf{G}^{-1} \neg p)) \quad (2.5)$$

has no equivalent in MTL under the signal semantics.

This formula can be understood as follows: it points to a time instant where slightly after date 1, and says that the previous non-empty event was a p . That it really finds a point close to date 1 is enforced by saying that it occurs after date 1 (because of the constraint ≥ 1 on the outermost modality $\mathbf{F}_{\geq 1}$) and is *not too far away* (at most one time unit apart) from a point where $\mathbf{G}^{-1} \neg p$ holds. This formula holds true along any of the $\mathcal{A}_{k,n}$, but fails to hold along any $\mathcal{B}_{k,n}$.

Remark. Notice that we do not mean that Formulas (2.5) and (2.4) are equivalent: they just share the property that they can separate (families of) models that MTL cannot.

To sum up the results of this section:

Theorem 5 ([BCM05, BCM10]). Under both the timed-word- and signal semantics,

- TPTL is strictly more expressive than MTL;
- MTL+Past is strictly more expressive than MTL;
- MITL+Past is strictly more expressive than MITL.

Remark. To the best of our knowledge, only the result concerning MITL+Past under the timed-word semantics was known [AH92b]. However, from [OW05], we get that there exists no algorithmic translation from MTL+Past to MTL interpreted on finite timed words, since in this setting, the latter logic is decidable while the former is not.

It must be noticed also that these results in the timed setting are in sharp contrast with the untimed case, where past-time modalities are known to add no expressive power to LTL [Kam68, GPSS80], even though they may bring succinctness [LMS02b].

2.2 Real-time model-checking with punctuality

As explained in the introduction of this chapter, undecidability of MTL satisfiability (and model checking) is tightly linked to punctual constraints: the undecidability proofs heavily rely on punctual constraints allowing to exactly reproduce sequences of events after a one-time-unit delay, and on the other hand, banning punctuality brings decidability.

We prove here that there are decidable timed temporal logics allowing punctual constraints. Instead of banning punctual intervals, our logics have restrictions on infinite intervals. We begin with a tour of decidable fragments of MTL. We then define the logics we consider and quickly address expressiveness questions. Finally, we propose model-checking algorithms under the timed-word and signal semantics.

2.2.1 Decidable fragments of MTL

Several decidable fragments of MTL have been defined in the literature. To begin with, let us mention that MTL itself has been proved decidable over *finite* runs in the timed-word semantics. In fact, as was noticed by Joël Ouaknine and James B. Worrell in 2005, the proof that we briefly sketched at the beginning of this chapter is only correct under the signal semantics.

In the timed-word semantics, while we require each event to occur exactly one time unit later, we cannot prevent *insertion errors*, *i.e.*, the rise of new events with no corresponding predecessor one time unit earlier. In the case of infinite runs, Joël Ouaknine and James B. Worrell proved that the recurrent-state problem for channel machines with *insertion errors* and emptiness test is undecidable [OW06a]. Such “faulty” channel

machines are precisely what the above encoding with MTL does. In the case of infinite words, this proves undecidability.

In the case of finite words, the above encoding cannot be adapted: finite words can encode reachability in faulty channel machines, but this problem can be proved decidable. Actually, MTL satisfiability (and model checking) over finite words is also decidable [OW05]: this can be proved by adapting the alternating-automata-based algorithm for LTL model-checking [Var94] to MTL. In the untimed case, the algorithm consists in turning a given LTL formula into a (linear-size) alternating automaton. Alternating automata extend non-deterministic automata with *conjunctive non-determinism*, allowing to fork several parallel executions on the input word. An execution of such an automaton on a given input word is a tree. Figure 2.4 displays an example of the algorithm (starting from formula $\mathbf{G}(a \Rightarrow \mathbf{F}b)$). The important remark to notice is that execution trees can be made *memoryless*, meaning that two occurrences of the same state can be the roots of the the same subtrees (the execution tree of the alternating automaton is then turned into a directed acyclic graph). It follows that there are finitely many different slices, so that (with some extra care w.r.t. the acceptance condition) alternating automata can be simulated by an (exponential-size) non-deterministic automata [MH84].

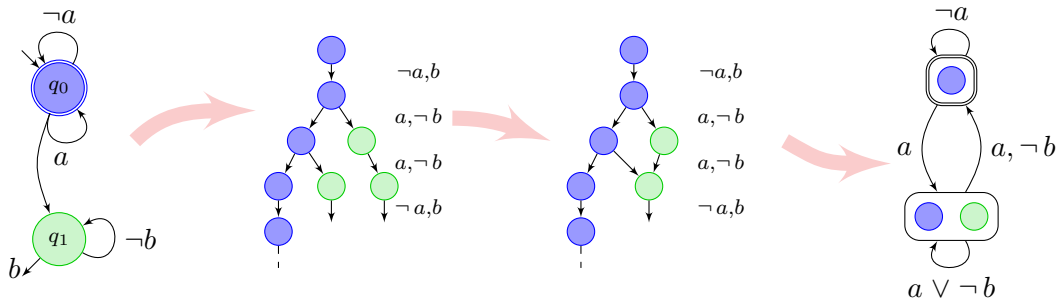


Figure 2.4: Simulating an alternating automaton with a non-deterministic one

In the timed case, the situation is more complicated. While any MTL formula φ can be turned into a one-clock timed alternating automaton \mathcal{B}_φ (see Equation (2.6) for an example of a transition formula), the corresponding non-deterministic automaton \mathcal{S}_φ has uncountably many states and infinite branching, since the clock may have uncountably many different values. However, each state of \mathcal{S}_φ , *i.e.* each set of pairs (s, v) where s is a state of \mathcal{B}_φ and v is a valuation of the clock of \mathcal{B}_φ , can be encoded as follows: the value of the clock is replaced with the region it belongs to, the resulting pairs (s, r) are sorted according to the fractional part of the clock, and the pairs sharing the same fractional parts are grouped together. For instance, a state C made of $(s_1, 0.3)$, $(s_2, 1.5)$, $(s_1, 2)$ and $(s_3, 2.3)$ will be represented as the sequence

$$H(C) = \{(s_1, c, r_4)\} \cdot \{(s_1, c, r_1), (s_3, c, r_5)\} \cdot \{(s_2, c, r_3)\}$$

where c is the name of the (unique) clock of \mathcal{B}_φ , and r_i represents the i -th 1-dimensional region ($r_0 = \{0\}$, $r_1 = (0, 1)$, ...) This encoding was proposed in [OW05], where it is

proved that two states having the same encoding are time-abstract bisimilar. Emptiness of the language accepted by the timed alternating automaton is then equivalent to emptiness of the quotient of \mathcal{S}_φ by the equivalence relation induced by H . The resulting non-deterministic automaton \mathcal{R}_φ now has countably many states, and finite branching. The important property of \mathcal{R}_φ is that it is a *well-structured transition system* [FS01]: the order on the set of states of \mathcal{S}_φ derived from the inclusion relation on sets of pairs (s, r) is a well-quasi-order and is compatible with the transitions of \mathcal{R}_φ (if there is a transition from a state, then there is a corresponding transition from a “sub-state”). From [FS01], it is decidable whether an accepting configuration can be reached from the initial state. This shows decidability of satisfiability of MTL over finite timed words.

Model checking can be handled quite similarly, by considering joint executions of the timed alternating automaton $\mathcal{B}_{\neg\varphi}$ and the automaton \mathcal{A} under checking: writing $\mathcal{S}_{\mathcal{A},\varphi}$ for the non-deterministic automaton corresponding to those joint executions, we again represent a state C of $\mathcal{S}_{\mathcal{A},\varphi}$ by projecting on regions and storing the order of fractional parts. If C is composed of $(s_1, 0.3)$, $(s_2, 1.5)$, $(s_1, 2)$ and $(s_3, 2.3)$ as states of $\mathcal{B}_{\neg\varphi}$ and $(\ell, \{x = 1.5, y = 0.4, z = 2.3\})$ (assuming \mathcal{A} has three clocks x , y and z), the corresponding encoding is

$$H(C) = \{(s_1, c, r_4)\} \cdot \{(s_1, c, r_1), (s_3, c, r_5), (\ell, z, r_5)\} \cdot \{(\ell, y, r_1)\} \cdot \{(s_2, c, r_3), (\ell, x, r_3)\}.$$

Again, it can be proved that two states with the same encoding can be proved to be timed-abstract bisimilar in $\mathcal{S}_{\mathcal{A},\varphi}$. The rest of the argument is the same, proving that MTL model checking is decidable over finite timed words by solving reachability of a downward-closed set of states in the well-structured transition system $\mathcal{R}_{\mathcal{A},\varphi}$.

In the case of infinite words, the most classical compromise to recover decidability is to let go of punctual constraints. MITL is the corresponding fragment, where punctual constraints are banned. This logic was defined in [AFH96], where it is shown to be less expressive than timed automata: for any MITL formula, there exists a timed automaton accepting exactly the same language. Intuitive ideas of the construction are sketched in Section 1.3.2, on page 26. Several different proofs have been obtained since then, by means of other fragments of MTL:

- the *Event-Clock Temporal Logic* (ECTL) has been introduced in [RS97, RS99] as a different restriction of MTL: it has unconstrained “until” and “since” modalities, and two extra modalities for imposing timing constraints on the *next* and *previous* occurrence of an event. For instance, $\triangleright_{\sim c}\varphi$ holds true at a given position whenever the time to the earliest time point in the future where φ holds satisfies the timing constraint $\sim c$.

This logic has been proved decidable by a transformation to *event-clock automata*, a restriction of timed automata where clocks are associated with events and reset when their associated event is encountered. Event-clock automata were defined in [AFH94] as a determinizable class of timed automata (hence closed under complement). Also, in the signal semantics, ECTL has been proved to be (effectively) equivalent to MITL+Past. Under the timed-word semantics, ECTL has been proved equivalent to MITL+Past $_{\sim}$ (in which timing constraints have the form $\sim c$ with $\sim \in \{<, \leq, \geq, >\}$) [Ras99].

- the *Quantitative Temporal Logic* (QTL), introduced and studied in [HR99, HR05], extends LTL with two modalities $\mathbf{F}_{<1}$ and $\mathbf{F}_{>-1}^{-1}$. This logic is proved to be expressively equivalent to MITL (with exponential blowup due to the binary encoding of integer constants), and decidable in polynomial space. The central concept for proving decidability is *timer formulas*, defined as

$$\text{Timer}(\varphi, \psi) = \mathbf{G}^{-1}(\psi \Leftrightarrow \mathbf{G}_{>-1}^{-1}\varphi) \wedge (\psi \Leftrightarrow \mathbf{G}_{>-1}^{-1}\varphi) \wedge \mathbf{G}(\psi \Leftrightarrow \mathbf{G}_{>-1}^{-1}\varphi)$$

(in other terms, $\text{Timer}(\varphi, \psi)$ is true whenever, all along the run, ψ holds if, and only if, φ has held true over the whole last time unit). With any QTL formula φ can be associated an LTL+Past formula ψ s.t. φ is satisfiable if, and only if, the conjunction of ψ with Timer-formulas is. By stretching the time-line, this in turn is proved equivalent to the satisfiability of a plain LTL+Past formula [HR05]. It must be noticed that this work considers the signal semantics with no restriction on finite variability.

After their remark in [OW05] that punctuality was not as harmful as expected for decidability, Joël Ouaknine and James B. Worrell introduced the fragment **SafetyMTL**, a fragment of MTL including punctuality and whose model-checking and satisfiability problems they prove decidable (though non-primitive-recursive and non-elementary, resp.) Compared to MTL, in **SafetyMTL** eventualities (*i.e.*, “until” modalities) must have bounded deadlines. This still offers a very interesting expressive power, as for instance any (unconstrained) safety property can be expressed in **SafetyMTL** (hence the name), as well as time-bounded-response-time properties.

It is proved in [OW05] that model checking **SafetyMTL** over infinite timed words is decidable. This relies on a translation of a **SafetyMTL** formula into a timed alternating automaton where all states are accepting (since eventualities are time-bounded, they are enforced by the restriction to non-Zeno timed words, and by enforcing the time bound on the outgoing transitions). Hence the complement of this timed alternating automaton has no accepting states, so that it accepts if, and only if, it reaches an empty configuration. This can be decided using similar techniques as for MTL over finite words.

Satisfiability of **SafetyMTL** was proved in [OW06b] (again assuming the timed-word semantics, and under the assumption that infinite timed words are time-diverging). This again goes via timed alternating automata, and uses the same region-abstraction as above to reduce to a well-structured transition system. Time-diverging runs of the timed alternating automaton correspond to *progressive* runs in the associated region automaton, and the set of configurations from which there is such a progressive run is a downward-closed set and can be characterized as a greatest fixpoint of a computable mapping.

A lower-bound on the complexity of satisfiability of **SafetyMTL** can be derived from [BMO⁺08], where we proved that termination of (all the runs of) a channel automaton with insertion errors and emptiness check is non-elementary (yet primitive recursive).

Under the signal semantics, the undecidability proofs for MTL apply to **SafetyMTL**.

With Patricia Bouyer-Decitre, Joël Ouaknine and James B. Worrell, we introduced two extra fragments of MTL:

- **BoundedMTL** is the fragment of **SafetyMTL** where also the “dual-until” modality is time-bounded. Under the assumption that infinite timed words are time-divergent, any formula in this logic only depends on a finite prefix of the execution it is evaluated on. Not surprisingly, the complexity of verification problems for this logic is much lower than that of **SafetyMTL**: we prove below that both model checking and satisfiability are **EXPSpace**-complete.
- **FlatMTL** is a restriction of **MTL** similar to **SafetyMTL**: unbounded “until” modalities are restricted to have **MITL** formulas as their left-hand-side formulas, and symmetrically, unbounded “dual-until” modalities may only have **MITL** formulas as their right-hand-side arguments.

Since we are mostly interested on model checking, we will consider the dual fragment **coFlatMTL**, which contains negations of **FlatMTL** formulas. Formally:

$$\text{coFlatMTL} \ni \varphi_p ::= \mathbf{p} \mid \varphi_p \vee \varphi_p \mid \varphi_1 \wedge \varphi_p \mid \varphi_p \mathbf{U}_I \varphi_p \mid \varphi_p \mathbf{U}_J \psi_p \mid \varphi_p \mathbf{R}_I \varphi_p \mid \psi_p \mathbf{R}_J \varphi_p$$

where I ranges over bounded intervals, J ranges over (bounded or unbounded) intervals, and ψ_p ranges over **MITL**. **coFlatMTL** is a very powerful logic for specifying properties of real-time systems, as it encompasses **LTL** and **BoundedMTL**, and is closed under \mathbf{G} . We prove below that **coFlatMTL** model checking is **EXPSpace**-complete. However, satisfiability of **coFlatMTL** is undecidable, as is readily proved from the undecidability proofs for **MTL** [AH92b, OW05]. The results for **FlatMTL** are dual (satisfiability is **EXPSpace**-complete, model checking is undecidable).

Figure 2.5 shows the relative expressiveness relations between these fragments of **MTL**.

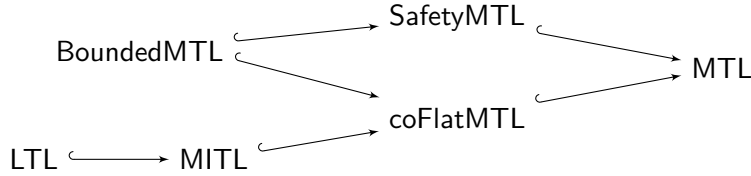


Figure 2.5: Fragments of **MTL**

2.2.2 **coFlatMTL** in the timed-word semantics

In this section, we present our model-checking algorithm for **coFlatMTL** over (infinite, time-diverging) timed words. Actually, we only present the algorithm for the weaker logic **coFlatMTL_{LTL}**, where the **MITL** formulas in the flatness condition are replaced with **LTL** formulas, as in [BMOW07]. We then briefly explain how to extend it to full **coFlatMTL**. An algorithm for satisfiability of **FlatMTL** formulas directly follows.

The algorithm is again an extension of the alternating-automata-based algorithm for **LTL** model checking: we turn the complement of the **coFlatMTL_{LTL}** formula φ into a (one-clock) alternating timed automaton $\mathcal{B}_{\neg\varphi}$, and consider the joint runs of $\mathcal{B}_{\neg\varphi}$ and the timed automaton \mathcal{A} under checking. Using the same encoding as above, we reduce the problem of finding an accepting joint run into the problem of finding an infinite

run in the associated (infinite-state) region automaton $\mathcal{R}_{\mathcal{A},\varphi}$. In order to reason about this infinite-state automaton, we introduce *channel automata*: channel automata are finite-state automata augmented with an (unbounded) FIFO channel. Our approach consists in simulating the region automaton $\mathcal{R}_{\mathcal{A},\varphi}$ on a channel automaton. For the simulation to work, we need to extend channel automata with two important features: renaming and occurrence testing. See Table 2 for some details on channel automata with renaming and occurrence testing (CAROTs).

The translation of an MTL formula³ into a one-clock alternating timed automaton follows the same ideas as for turning an LTL formula into an alternating automaton: each subformula corresponds to a state of the automaton, and the transition relation for “constrained-until” modality looks as follows:

$$\delta(\varphi_1 \mathbf{U}_I \psi_2, \sigma) = \left((\psi_1 \mathbf{U}_I \psi_2 \wedge x.\psi_1) \vee (x \in I \wedge x.\psi_2) \right) \quad (2.6)$$

where $x.\psi_1$ indicates a transition along which clock x is reset. The accepting states are those corresponding to “dual-until” subformulas.

Because the initial formula is in $\text{FlatMTL}_{\text{LTL}}$ ⁴, this automaton enjoys crucial properties besides *linearity* (a.k.a. *weakness*), which is classical when going from LTL to alternating automata. These properties ensure a kind of *progress* during the run of the automaton:

- first, self-loops are non-resetting (while any other state change involves resetting the clock);
- second, after a certain amount of time spent in a state ℓ , there cannot be a conjunctive transition from ℓ to itself and to a state involving punctuality. This follows from the flatness of the formula;
- last, any execution tree of this automaton can be sliced in such a way that *active* slices (*i.e.*, slices containing states where the clock value is less than the maximal constant M of the automaton) have short total duration.

We now explain how we encode the joint executions of $\mathcal{B}_{\neg\varphi}$ and \mathcal{A} . As previously, we consider the region abstraction, keeping only the integer values of the clocks and the order of their fractional parts. Each configuration of the joint execution of $\mathcal{B}_{\neg\varphi}$ and \mathcal{A} is a finite ordered list of items of the form (ℓ, x, v) where

- either ℓ is a state of $\mathcal{B}_{\neg\varphi}$, in which case x is the name of the unique clock of \mathcal{B} and v is its integer part (or the special symbol ∞ if this value is larger than the maximal constant);
- or ℓ is a state of \mathcal{A} (in which case it must be the same state for each similar items of the same configurations), x is the name of a clock of \mathcal{A} and v is its integer value (or ∞).

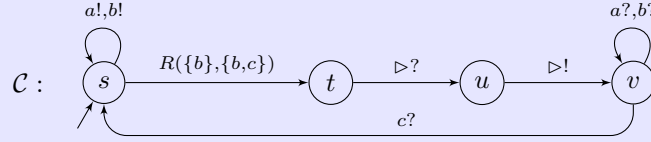
³This part of the construction applies to the whole logic MTL.

⁴Bearing in mind that we do $\text{coFlatMTL}_{\text{LTL}}$ model checking, so that our aim is to check that no run of a given timed automaton satisfies the *negation of the input $\text{coFlatMTL}_{\text{LTL}}$ formula*, which is a $\text{FlatMTL}_{\text{LTL}}$ formula.

A *channel automaton with renaming and occurrence testing* (CAROT) is a tuple $\mathcal{C} = \langle S, s_0, \Sigma, \Delta, F \rangle$ where S is a finite set of states containing in particular the initial state s_0 and the set F is final states, Σ is a finite alphabet and $\Delta \subseteq S \times \text{Op} \times S$ is the transition relation. Here, Op is the set of operations to be performed on the channel when firing transitions. Namely:

- $\sigma?$ and $\sigma!$ are classical operations for popping (when possible) and pushing letter σ from/onto the channel;
- $zero(\sigma)$ is a guard preventing the transition to be firable when the channel contains occurrence(s) of σ ;
- $R(S, S')$, where $S \subseteq \Sigma$ and $S' \subseteq \Sigma \cup \{\varepsilon\}$, non-deterministically renames occurrences of letters from S into letters from S' (where replacing by ε corresponds to deletion).

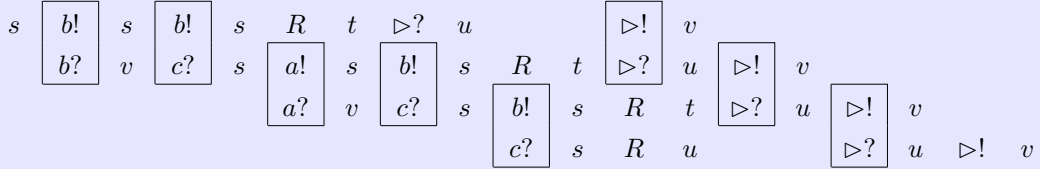
We use a special letter \triangleright for counting the number of *cycles* of a computation: formally, this letter is written in the channel at the beginning of the computation, and it is rewritten (at the tail of the channel) each time it is read, in such a way that the channel always contains exactly one copy of \triangleright (except when it has just been read).



The important property that we use is the following:

Theorem ([BMOW07]). *The cycle-bounded reachability problem for CAROTs is decidable in polynomial space in the size of the channel automaton and in the value of the cycle bound.*

The main idea behind this result is to *stack* the sequences of configurations of each cycles, with matching pushing operations on the channel with popping operations at the next cycle, as depicted below:



From a finite computation, we get a finite word over Σ^k , where k is the number of cycles. The algorithm consists in sliding a window along this word (see below) and checking that it corresponds to a correct computation of the CAROT. This may require guessing renaming rules to be checked later along the run (for instance, in the right-most sliding window below, the b written on the third cycle is renamed as a c by the renaming operation occurring just after the b is written), but only involves polynomial space.

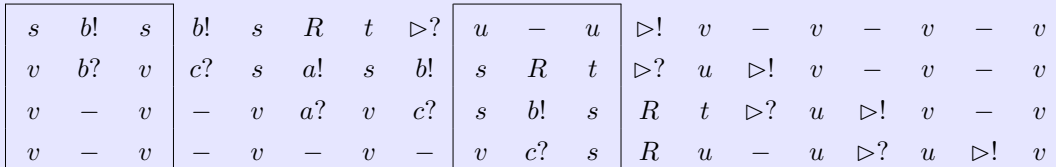


Table 2: Channel automata with renaming and occurrence testing

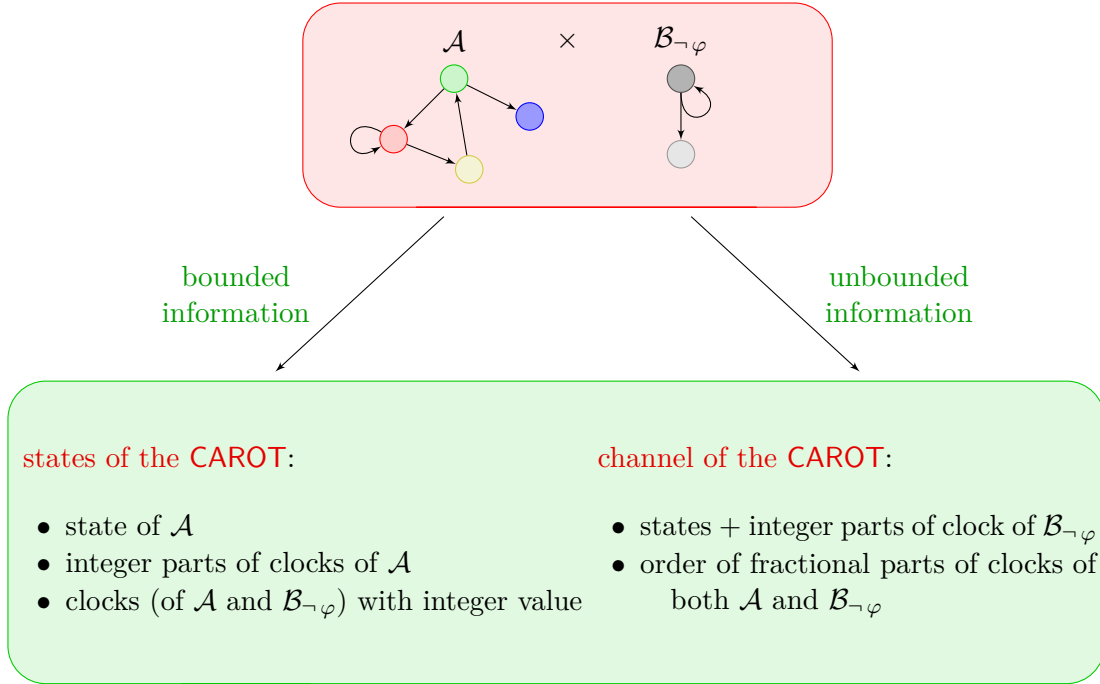


Figure 2.6: Encoding of joint executions of $\mathcal{A} \times \mathcal{B}_{\neg\varphi}$ on CAROTs

The order in this list is according to the fractional parts of the corresponding clocks. Encoding this in a CAROT $\mathcal{C}_{\mathcal{A},\varphi}$ is achieved as follows: the sequence of items above is stored on the channel, following the aforementioned order. The discrete state of the CAROT only contains the current state and region of \mathcal{A} , and the special cases of items of $\mathcal{B}_{\neg\varphi}$ (namely, items where the clock has integer value or value larger than the maximal constant, or where the state corresponds to an LTL formula). This is depicted on Figure 2.6. This way, the state-space of the CAROT is finite, but the channel allows to encode all the required information about a configuration. Renaming is used for transitions resetting clocks, allowing us to remove the corresponding items from the channel. Occurrence testing are needed when simulating transitions of $\mathcal{B}_{\neg\varphi}$, for over-approximating the configuration of $\mathcal{B}_{\neg\varphi}$ for which we have to compute successors.

The acceptance condition is encoded as in the classical Miyano-Hayashi construction, keeping track of those branches that *recently* hit an accepting state of $\mathcal{B}_{\neg\varphi}$.

In the end, we have

Proposition 6. *There is an accepting run in $\mathcal{C}_{\mathcal{A},\varphi}$ iff $\mathcal{A} \models \varphi$.*

The above properties of the timed alternating automata obtained from $\text{FlatMTL}_{\text{LTL}}$ formulas enforce the following property: if $\mathcal{C}_{\mathcal{A},\varphi}$ has an accepting run, then there is one that can be split into two parts:

- the first part is cycle-bounded (where the number of cycles is exponential in the sizes of \mathcal{A} and φ),
- the second part only involves states of $\mathcal{B}_{\neg\varphi}$ corresponding to LTL formulas.

The algorithm then consists in running the algorithm for checking the cycle-bounded reachability in CAROTs (see Table 2). Along the second part, the content of the channel only contains informations about \mathcal{A} , since configurations of $\mathcal{B}_{\neg\varphi}$ only involve LTL formulas, which are not written to the channel. This way, the CAROT can be handled as a finite state automaton. In the end, the whole algorithm can be run using exponential space.

Extending this algorithm to full coFlatMTL can be achieved by turning the MITL formulas involved in φ into equivalent non-deterministic timed automata, and simulating joint executions of these timed automata with \mathcal{A} when $\mathcal{B}_{\neg\varphi}$ reaches the corresponding MITL state. In the second phase of the algorithm, we can again bound the content of the channel in the same way as above (except that the channel still contains items from $\mathcal{B}_{\neg\varphi}$ issued from these MITL automata), which again provides an exponential-space algorithm.

Hardness in EXPSPACE can be proved by encoding the acceptance of a word by a 2^n -space-bounded Turing machines into the satisfiability of a BoundedMTL formula. We do not give the full reduction here, but only two examples of (families of) formulas that require doubly-exponential variability. These formulas are based on the following basic formula:

$$\varphi_d : (a \Rightarrow \mathbf{F}_{=1} (a \wedge \mathbf{X}_{\leq 1} b)) \wedge (b \Rightarrow \mathbf{F}_{=1} (a \wedge \mathbf{X}_{\leq 1} b)).$$

This formula forces to have at least twice as many alternations of a and b in a time-unit interval compared to the previous time unit. It then suffices to enforce this property for 2^n consecutive time units to get the desired family of formulas:

$$\varphi_n : a \wedge \varphi_d \wedge \mathbf{G}_{\leq 2^n} \varphi_d.$$

In the end:

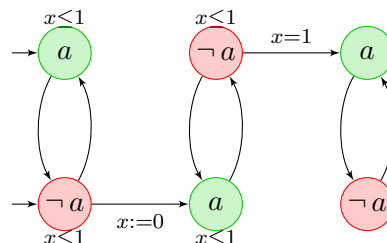
Theorem 7 ([BMOW07]). *The model-checking problem for coFlatMTL (and BoundedMTL) under the timed-word semantics is EXPSPACE-complete.*

Both the algorithm and the hardness proof can easily adapted to handle satisfiability for FlatMTL and BoundedMTL:

Theorem 8 ([BMOW07]). *The satisfiability problem for FlatMTL (and BoundedMTL) under the timed-word semantics is EXPSPACE-complete.*

2.2.3 coFlatMTL in the signal semantics

The technique presented above cannot cope with the signal semantics: by relying on CAROTs, it is inherently *sequential*. We propose an alternative technique, which borrows some ideas from [HR05] and in a sense resembles the *tableau* technique for LTL, extended to continuous-time, finitely-varying signals. We only sketch here the algorithm for FlatMTL satisfiability. Model checking can then be solved using a result



from [HRS98, Ras99] that the language of any timed automaton is a *projection* of the language satisfying an MITL formula. For instance, the timed automaton above recognises those signals where periods of a and $\neg a$ alternate, and where within the first two time units, there are two a -intervals whose starting dates are exactly one time unit apart. This automaton is a classical example of a non-complementable timed automaton, hence its language is not expressible in MITL. Taking advantage of projection, we can characterize it as

$$\exists p, q. \text{one}(p) \wedge \text{one}(q) \wedge \mathbf{F}(p \text{ Starts } a) \wedge \mathbf{F}(q \text{ Starts } a) \wedge \mathbf{F}_{\leq 1}(p \Leftrightarrow (\mathbf{F}_{\leq 1} q \wedge \mathbf{F}_{\geq 1} q))$$

where $\text{one}(X)$ is an LTL formula expressing that there is exactly one time point where X holds, and $X \text{ Starts } Y$ is an LTL formula saying that X holds at a point where Y rises⁵. The above formula imposes to have p and q hold at starting points of a -intervals, and takes advantage of the uniqueness of q to express punctuality using MITL. This provides us with a way of reducing model checking coFlatMTL to satisfiability of FlatMTL, since the latter subsumes MITL (the extra quantification then being part of the satisfiability problem).

Let us first quickly review the tableau technique for a formula φ of LTL. We consider an extended set of subformulas of φ , called the *closure* of φ , which is the smallest set of formulas containing φ and satisfying several rules such as:

$$\text{if } \psi_1 \mathbf{U} \psi_2 \in \text{cl}(\varphi), \text{ then } \varphi_1, \varphi_2 \text{ and } \mathbf{X}(\psi_1 \mathbf{U} \psi_2) \text{ are in } \text{cl}(\varphi).$$

A Hintikka sequence is then an infinite sequence of maximally consistent subsets of $\text{cl}(\varphi)$ in which each constraint of the form $\mathbf{X}\zeta$ in any element of this sequence implies the presence of ζ in the next element. It is also required that any eventuality encountered at some point is later fulfilled. Satisfiability of φ is equivalent to the existence of such a Hintikka sequence; this can be verified by checking non-emptiness of the *tableau* of φ , which is the automaton generating the Hintikka sequence of φ .

We extend this approach to FlatMTL under the signal semantics. In that setting, the closure of a formula is defined in the same way as for LTL formula, with extra rules for constrained modalities, such as

$$\text{if } \mathbf{F}_I \psi \in \text{cl}(\varphi), \text{ then } \mathbf{F}_{I-1} \psi \text{ are in } \text{cl}(\varphi)$$

⁵Formally, this may require past-time modalities, or considering a point slightly before the rise of Y . We omit those details for the sake of readability.

where $I - 1$ is obtained by shifting I by one unit to the left, and intersecting with $\mathbb{R}_{\geq 0}$.

The notion of Hintikka sequences is extended to the continuous framework: a *closure labelling* f is a mapping from $\mathbb{R}_{\geq 0}$ to $2^{\text{cl}(\varphi)}$ satisfying the same kind of rules as for Hintikka sequences, augmented with special rules for constrained modalities such as

$$\text{if } \mathbf{F}_I \psi \in f(s) \text{ then } \mathbf{F}_{I-1} \psi \in f(s+1) \text{ or } \psi \in f(s+\delta) \text{ for some } \delta \in (0, 1] \cap I.$$

A carefully crafted list of rules entail that satisfiability of a FlatMTL formula is equivalent to the existence of a closure labelling for it.

We now have to design an algorithm for deciding the existence of such a closure labelling. The ideas resemble the timed-word case in the following two respects:

- an important property that we get from our initial formula being in FlatMTL is that we can restrict to “simple” closure labellings, in the sense that punctual formulas may only appear in finitely many bounded-size intervals of the closure labelling, and the closure labelling has bounded variability (all the bounds being exponential in the input).
- we will *stack* several shifted copies of the closure labelling, hence working with a *multi-track* closure labelling. More precisely, given a closure labelling g , we consider the signal $T: \mathbb{R}_{\geq 0} \rightarrow (2^{\text{cl}(\varphi)})^k$ (where k will be the total size of the “punctual parts” mentioned at the previous point). The notion of closure labelling for this kind of signal is a bit modified, in that it now contains two kind of rules: *horizontal rules* are the same rules as for classical closure labelling, but restricted to non-punctually-constrained modalities, while *vertical rules*, relating the same point on different tracks of T , will handle both punctual and non-punctual constraints.

We have now reduced our problem to finding a bounded-variability multi-track closure labelling with slightly different rules. The important point to notice is that horizontal rules only involve non-punctual requirements: as remarked in [HR05], this kind of requirements can be expressed in LTL+Past up to *stretching equivalence*. Following this remark, we express our horizontal and vertical rules in LTL+Past: vertical rules simply correspond to local constraints on the different tracks of the signal, while horizontal rules are encoded using a set of extra `tick` atomic propositions for approximating time lapses. Past-time modalities are needed *e.g.* for formulas of the form $\mathbf{G}_{=1} \psi$: checking that this formula holds at a point on track i is achieved by checking that ψ holds on track i until the next `tick` and has been holding on track $i+1$ since the previous `tick`.

Figure 2.7 depicts the reduction: in this drawing, if the multi-track signal satisfies the LTL+Past formula, then we can come up with a single-track signal (following the white line) satisfying the FlatMTL formula under study (provided that each black interval, involving punctual constraints, is stretched to last exactly one time unit).

This transforms the satisfiability problem for a FlatMTL formula to the satisfiability problem for an exponential-size LTL+Past formula (over $\mathbb{R}_{\geq 0}$), which is PSPACE-complete [Rey10]. Since our encoding involves an exponential blowup, the whole algorithm uses exponential space. Using similar techniques as in the timed-word semantics, we can prove that this is optimal, in the sense that satisfiability of FlatMTL can encode the halting problem of a 2^n -space-bounded Turing machine. Summarizing these results, we get:

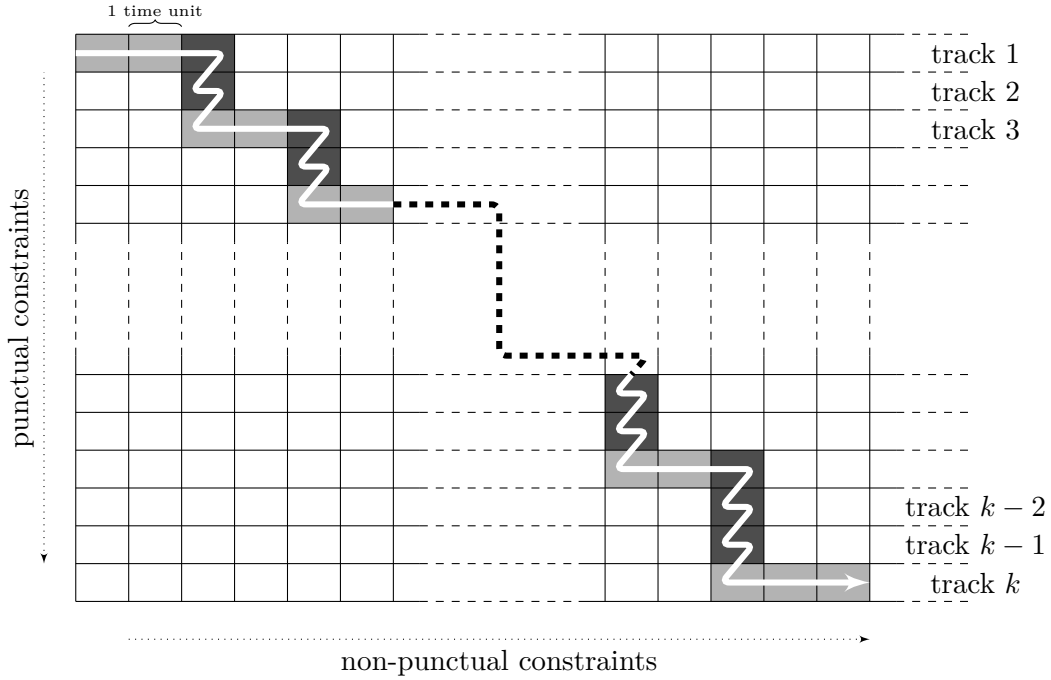


Figure 2.7: Schematic representation of a multi-track closure labelling

Theorem 9 ([BMOW08]). *The satisfiability problem for FlatMTL (and BoundedMTL) and the model-checking problem for coFlatMTL (and BoundedMTL) under the signal semantics are EXPSPACE-complete*

2.3 Robust model-checking

While in the previous section we struggled for checking punctual constraints, in the coming one we advocate for taking imprecisions into account when model checking timed automata. This has more practical motivations, as timed automata are governed by a mathematical semantics, assuming infinite precision and speed, while computers are digital and imprecise. We describe the approach in a first part, and compare it with related works that have been undertaken for coping with these imprecisions. We then present our *robust-model-checking* algorithm for safety properties first, and its extensions to several temporal logics.

2.3.1 Implementability of timed automata

Timed automata are a mathematical model used for representing real-time embedded systems. Their success is in a large part due to the fact that they propose an interesting balance between expressiveness and efficiency of verification algorithms (and tools). However, the properties that we check of these models may be lost during the implementation

phase, because of the discrepancies between the mathematical model and the physical devices they will eventually be implemented and run on.

Several proposals have been studied to take this into account and come with more faithful semantics for timed automata. In 1997, Vineet Gupta, Thomas Henzinger, and Radha Jagadeesan introduced the *tube semantics* for timed automata: given a metrics on finite runs, a tube is a non-empty open set of runs. A tube is generated by a timed automaton if it contains a dense subset of runs that are generated by this timed automaton (for the classical semantics). A *robust timed automaton* is a timed automaton equipped with this *tube semantics*. Intuitively, this restricts the set of accepted traces to those that would be accepted even if the timing would slightly change. Emptiness (under the tube semantics) can be checked by considering the *interior automaton*, which is a timed automaton obtained from the original automaton by considering strict versions of the constraints and adding some non-determinism in the updates. Emptiness of the interior automaton is checked using its region automaton. This can be achieved in PSPACE [GHJ97].

Another approach was proposed by Anuj Puri in 1998. The approach is completely different, since it *enlarges* the language instead of restricting it: given a timed automaton \mathcal{A} and a value $\varepsilon > 0$, the ε -drifting automaton is the hybrid automaton \mathcal{A}_ε obtained by letting the slope of all clocks take values in $[1 - \varepsilon, 1 + \varepsilon]$. For any positive ε , \mathcal{A}_ε has more behaviours than \mathcal{A} . The set of reachable configurations in the *drifting semantics* is the intersection of the sets of reachable configurations for any $\varepsilon > 0$. This set of reachable configurations can be computed in polynomial space, using an extension of the region automaton [Pur98]. Our approach, detailed below, reuses many of the ideas of this work.

Finally, let us mention the *probabilistic-semantics* approach proposed in 2007 by Christel Baier, Nathalie Bertrand, Patricia Bouyer-Decitre, Thomas Brihaye and Marcus Größer: the semantics is based on the measure of the set of runs satisfying a given property among the whole set of runs. Again, it restricts the set of behaviours of the automaton, by discarding the unlikely ones. The problem of deciding whether the set of runs of a one-clock timed automaton satisfying a given LTL formula has measure 1 is proved decidable, by considering the region automaton without *unlikely* transitions as a finite Markov chain [BBB⁺08]. In [BBBM08], we extended the approach to the *quantitative* measure of the set of runs satisfying an ω -regular property (still in a one-clock timed automaton, with some extra technical restrictions), also by reducing the region automaton into a finite Markov chain.

My main contributions to the problem started during my post-doc in the team of Jean-François Raskin in Brussels, and is based on the approach of [DDR04]: this paper proposes a new semantics of timed automata based on a simple model of a CPU on which the timed automaton would execute: the clock on which time is measured has a finite frequency f_{clock} (meaning that it is incremented by Δ_{clock} every Δ_{clock} time units, where $\Delta_{\text{clock}} = 1/f_{\text{clock}}$), and the CPU runs the following cycle with a finite frequency f_{CPU} :

- read the value of the clock;
- compute the values of the guards;

- perform one of the available transitions.

The resulting semantics, which we call *program semantics*, is parametrised by Δ_{clock} and $\Delta_{\text{CPU}} = 1/f_{\text{CPU}}$. The *implementability* problem now asks whether there exists positive values of these parameters under which all the executions of the automaton (under this modified semantics) satisfy a given property.

The presence of two parameters and the inherent *non-smoothness* of this semantics make it hard to work with. In order to simplify the problem, Raskin *et al.* proposed another alternative semantics, called the *enlarged semantics*: it consists in enlarging all timing constraints by a (single) parameter Δ . For instance, a timing constraints requiring that the value of a clock c lies in $[3, 5]$ would be replaced by $x \in [3 - \Delta, 5 + \Delta]$, thus modelling the fact that the guard may be checked a bit earlier or later than expected. Under a simple condition linking Δ with Δ_{clock} and Δ_{CPU} , it was proved that $\llbracket \mathcal{A} \rrbracket_{\Delta_{\text{CPU}}, \Delta_{\text{clock}}}^{\text{prog}} \subseteq \llbracket \mathcal{A} \rrbracket_{\Delta}^{\text{enlarge}}$, where $\llbracket \mathcal{A} \rrbracket$ represents the set of all the executions of the timed automaton \mathcal{A} under the corresponding semantics⁶. This makes the enlarged semantics a good tool for checking (linear-time) properties of possible implementations of timed automata. Notice that because of the inclusion above not being an equality, the method is not complete. The *robust-model-checking* problem is then the following: given a linear property φ and a timed automaton \mathcal{A} , does there exist a value for Δ for which $\llbracket \mathcal{A} \rrbracket_{\Delta} \subseteq \llbracket \varphi \rrbracket$?

2.3.2 Robust verification of safety properties

A simple yet natural property to consider first is safety: we are given a set B of states, and want to prove that they are not reached under the enlarged semantics, for some positive value of the parameter Δ .

The first point to notice is that the enlarged semantics enjoys the *faster-is-better* property: if the above property holds for some $\Delta_0 > 0$, then it also holds for $0 < \Delta \leq \Delta_0$, since obviously we have $\llbracket \mathcal{A} \rrbracket_{\Delta} \subseteq \llbracket \mathcal{A} \rrbracket_{\Delta_0}$. As a consequence, writing $\text{Reach}_{\Delta}(\mathcal{A}, \ell_0)$ for the set of configurations of \mathcal{A} reachable from ℓ_0 under the enlarged semantics, the limit

$$\text{Reach}^{\text{enlarge}}(\mathcal{A}, \ell_0) = \bigcap_{\Delta > 0} \text{Reach}_{\Delta}(\mathcal{A}, \ell_0)$$

is well-defined. Moreover, using topological arguments, it can be proved that

$$\text{Reach}^{\text{enlarge}}(\mathcal{A}, \ell_0) \cap B \neq \emptyset \quad \Leftrightarrow \quad \exists \Delta_0 > 0. \text{Reach}_{\Delta_0}(\mathcal{A}, \ell_0) \cap B \neq \emptyset,$$

thus reducing our original parametrised problem to a non-parametrised one. The set $\text{Reach}^{\text{enlarge}}(\mathcal{A}, \ell_0)$ is the set of configurations that can be reached in the enlarged semantics for any positive enlargement: this way, it does not include *border of regions* of width Δ . Still, as exemplified on Figures 2.8 to 2.10, $\text{Reach}^{\text{enlarge}}(\mathcal{A}, \ell_0)$ may differ from $\text{Reach}(\mathcal{A}, \ell_0)$. Notice that the automaton \mathcal{A}_2 (*i.e.*, where α is replaced by 2 in the transition to B) is not robust for the safety property consisting in avoiding state B , since the configuration $(\ell_2, x = 0 \wedge y = 2)$ is in $\text{Reach}^{\text{enlarge}}(\mathcal{A}_2, \ell_0)$. On the other hand, the automaton \mathcal{A}_3 can be safely implemented (on a sufficiently competitive hardware).

⁶From this point on, we will not mention the *program semantics* anymore, and will simply write $\llbracket \mathcal{A} \rrbracket_{\Delta}$ for the Δ -enlarged semantics.

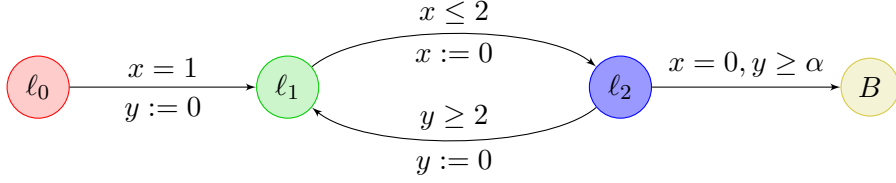


Figure 2.8: A timed automaton \mathcal{A}_α

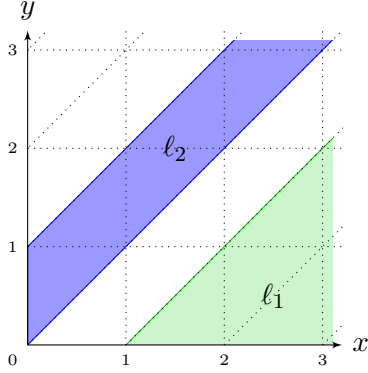


Figure 2.9: $\text{Reach}(\mathcal{A}_\alpha, \ell_0)$

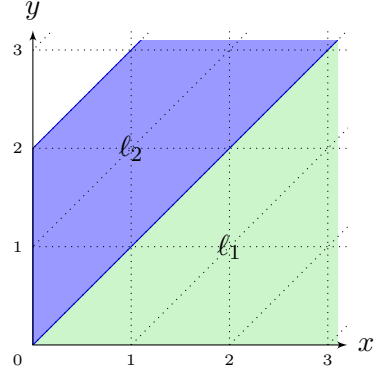


Figure 2.10: $\text{Reach}^{\text{enlarge}}(\mathcal{A}_\alpha, \ell_0)$

Robust model checking now amounts to computing $\text{Reach}^{\text{enlarge}}(\mathcal{A}, \ell_0)$. We prove that the set of configurations computed by Puri’s algorithm for drifting clocks (see Section 2.3.1) is precisely our set $\text{Reach}^{\text{enlarge}}(\mathcal{A}, \ell_0)$. The idea of is as follows: since we consider the set of configurations that can be reached for arbitrarily small Δ , the newly reachable configurations (besides those reachable in the classical semantics) will be reached by iterating cycles, thus accumulating imprecisions in the relative values of the clocks. The algorithm, displayed at Algorithm 1, works as follows: it first compute reachable configurations in the classical semantics, and then adds each cycle (of the region automaton) that skims this set of reachable configurations.

Back to our example of Figure 2.8: there is a cycle around region⁷ ($\ell_1, x \in [0, 1] \wedge y = 0$), visiting region ($\ell_2, x = 0 \wedge y \in [1, 2]$). No matter how small Δ is, this cycle can be entered from the set of reachable configurations in the usual semantics (*e.g.* from the right of region ($\ell_1, x \in [0, 1] \wedge y = 0$)) and iterated in order to reach the whole of this region.

The proof that the set J^* computed by Algorithm 1 equals $\text{Reach}^{\text{enlarge}}(\mathcal{A}, \ell_0)$ follows Puri’s proof for the case of clock drifts. It consists in proving both inclusions:

- the main ingredient for proving soundness (*i.e.*, $J^* \subseteq \text{Reach}^{\text{enlarge}}(\mathcal{A}, \ell_0)$) is the fact that if the timed automaton has a run with the same initial and final configuration, then the neighbourhood of this configuration (within its region) can be reached.

⁷In the setting of robustness, we consider a small variant of the notion of regions, namely *closed regions*—see Remark 2.3.2 for a note on this. We keep the notation $[v]$ for the tightest closed region containing valuation v .

Algorithm 1: Computes $\text{Reach}^{\text{enlarge}}(\mathcal{A}, \ell_0)$ for a (closed) timed automaton \mathcal{A} .

Data: A timed automaton $\mathcal{A} = \langle \mathcal{S}, \mathcal{X}, \text{cond}, \text{inv} \rangle$, an initial state $s_0 \in S$.

Result: The set J^* , which we prove equals $\text{Reach}^{\text{enlarge}}(\mathcal{A}, s_0)$

begin

```

1   |   Construct the region automaton  $G = (R_{\mathcal{A}}, T_G)$  of  $\mathcal{A}$  ;
2   |   Compute the set  $\mathbf{C}(G)$  of simple cycles of  $G$  ;
3   |    $J^* \leftarrow \text{Reach}(G, [q_0])$  ;
4   |   while for some  $p = p_0 p_1 \dots p_k \in \mathbf{C}(G)$ ,  $[p_0] \not\subseteq J^*$  and  $J^* \cap [p_0] \neq \emptyset$  do
5   |   |    $J^* \leftarrow J^* \cup [p_0]$  ;
6   |   |    $J^* \leftarrow \text{Reach}(G, J^*)$  ;
7   |   return  $J^*$  ;

```

end

From this we can prove that if there is a cycle in the region automaton, then all the configurations in the visited regions are reachable in the timed automaton.

- correctness ($\text{Reach}^{\text{enlarge}}(\mathcal{A}, \ell_0) \subseteq J^*$) is achieved by building a run in the classical semantics that closely follows a given run in the enlarged semantics. This proof is quite technical, involving parametric DBMs in order to keep track of the amount of imprecision accumulated along a run.

Remark. As for Puri's proof [Pur98], our proof only holds of a subclass of timed automata, in which

- no clock constraint involves strict inequalities;
- all clocks are bounded by some maximal constant M ;
- all clocks are reset (at least once) along any cycle in the region graph.

The first condition is quite natural in the setting of robust model checking, as a strict inequality enlarged by Δ is weaker than the corresponding closed inequality enlarged by $\Delta/2$. Because of this condition, we only consider closed region in our constructions.

The second condition is also not very restrictive, as it can be achieved by considering extra copies of each states, in which the clocks whose values exceed M are just irrelevant (we just have to remember that they are larger than the maximal constant M).

While the first two conditions are rather harmless, the third one is a bit stronger. Even though it encompasses strongly non-Zeno timed automata, it excludes a class of timed automata that it may be useful in practice. In recent (and yet unpublished) works with Patricia Bouyer-Decitre and our PhD student Ocan Sankur, we have dropped this restriction, thanks to a different algorithm relying on CAROTs (see Section 2.2.2 for more on CAROTs, and the next section for some insight on how they can be used for robust model checking).

Notice that as a side result, we get that under the above restrictions, clock drifts and guard enlargement produce the same kind of perturbations to the timed automaton,

in the sense that the set of configurations that are made reachable by clock drifts or guard enlargement, no matter how small, are the same. Actually, we even proved in [DDMR04, DDMR08] that the same result if both sources of imprecision are combined.

In terms of complexity, Algorithm 1 runs in exponential time, and can be implemented to run on-the-fly, thus using at most polynomial space. By adapting a reduction from [CY92], encoding the acceptance problem for a linear-bounded Turing machine, we can prove that this is essentially optimal.

Theorem 10 ([DDMR04, DDMR08]). *Robust model checking safety properties is PSPACE-complete.*

2.3.3 Robust verification of LTL and coFlatMTL

Based on these encouraging results for robust model checking, we decided to push the technique further and to do robust model checking for temporal logics. The approach presented above is geared towards trace inclusion, so that only linear-time temporal logics are meaningful in this setting.

With Patricia Bouyer-Decitre and Pierre-Alain Reynier (her PhD student at that time), we started with LTL. The approach consisted in applying the classical LTL-to-Büchi transformation. Checking that all the runs in the enlarged semantics do satisfy the LTL formula is then equivalent to checking that the product of the timed automaton \mathcal{A} under checking with the automaton $\mathcal{B}_{\neg\varphi}$ for the negation of the LTL formula is empty, *i.e.*, satisfies a co-Büchi condition.

Such conditions can be checked on the region automaton. However, the classical region automaton only represents the classical semantics of timed automata. Based on the results of the previous section, we define the *extended region automaton* $\mathcal{R}^*(\mathcal{A})$, which contains extra transitions on top of the classical region automaton. Those extra transitions correspond to the extra behaviours that we have characterized previously: more precisely, there must be an extra transition between two regions when the intersection of those (closed) regions is non-empty, and the target region belongs to a cycle.

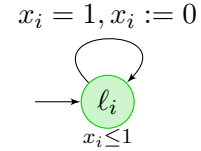
It is easily seen that Algorithm 1 amounts to compute the set of reachable regions in this extended region automaton. We proved that $\mathcal{R}^*(\mathcal{A})$ is also correct for checking co-Büchi conditions, in the following sense: a timed automaton satisfies a co-Büchi condition (defined in terms of the states of the automaton) under the enlarged semantics if, and only if, the extended region automaton satisfies the same co-Büchi condition. This technical result provided us with an algorithm for robust model checking LTL properties. The algorithm is easily seen to be implementable using polynomial space.

Theorem 11 ([BMR06]). *LTL robust model checking is PSPACE-complete.*

Extending this approach of robust model checking to quantitative properties is more involved. To this aim, we call upon channel automata, with the following intuition:

in the same way as we used CAROTs to simulate the behaviour of timed automata (under the classical semantics), we can use CAROTs to simulate the enlarged semantics of timed automata, assuming a fixed enlargement parameter Δ of the form $1/n$: this can be achieved by writing n copies of a special letter Δ to the channel, and inserting clocks in between during the simulation. Each time a Δ -symbol is read, it is rewritten immediately on the tail of the channel. In order to simulate the *enlarged* semantics, we may have to anticipate or delay some transitions: given a guard $x \geq 2$, transformed into $x \geq 2 - \Delta$ in the enlarged semantics, the corresponding transition can be fired before clock x reaches the head of the channel: if x has integer value 1 and is close to the head of the channel (*close* meaning that there may be one occurrence of Δ , as well as other clocks, above x on the channel), then the transition can already be taken in the enlarged semantics. Our simulating CAROT will allow this transition to be taken at any time, and store the information that it has to check the presence of x among the next symbols read from the channel. Guards of the form $x \leq 2$, enlarged to $x \leq 2 + \Delta$, are handled analogously: the CAROT will store the information whether clock x has been seen *recently*, and will allow the transition if it is the case.

In order to prove correctness of this construction (in the sense that the CAROT really simulates the timed automaton under study with the Δ -enlarged semantics), we introduce an intermediary step in the encoding: we transform the input timed automaton into a *network of timed automata* under the standard semantics: for a given $\Delta = 1/n$, we introduce n small timed automata $(\mathcal{B}_i)_{0 \leq i < n}$, as the one depicted on the right. Each \mathcal{B}_i has one clock x_i , whose role is to delimit Δ -time-unit-long intervals of time. Thanks to these clocks, we can simulate guard enlargement with the classical semantics of timed automata, in such a way that an MTL formula holds robustly in \mathcal{A} iff it holds in the product of \mathcal{A} with a family $(\mathcal{B}_i)_i$ (assuming initial valuation $v(x_i) = i/n$ for the new clocks). For instance, a guard of the form $x \geq c$ is replaced with



$$(x \leq c + 1) \wedge \left(x > c \Rightarrow \bigwedge_{0 \leq i < n} \langle x \rangle \leq x_{i+1} < x_{i-1} \right) \quad (2.7)$$

where $\langle x \rangle$ is the fractional part of clock x . While fractional parts are not allowed (syntactically) in guards, it can be encoded by associating with each clock an extra clock recording its fractional part (reset at the same time, as well as when it reaches 1). This transformation can be proved to be equivalent to classical guard enlargement, in the following sense: for $n \geq 3$, $\llbracket \mathcal{A} \rrbracket_{\frac{1}{n}} \subseteq \llbracket \mathcal{B}^n \rrbracket \subseteq \llbracket \mathcal{A} \rrbracket_{\frac{2}{n}}$, where \mathcal{B}^n is the product of the \mathcal{B}_i 's with the automaton obtained from \mathcal{A} by transforming guards as in (2.7).

Now, the region automaton corresponding to this network can be simulated on a CAROT. The important thing to notice is that the CAROT itself only depends on \mathcal{A} , and not on the value of n : indeed, as the extra clocks all play a symmetric role, they can be represented with the same letter on the channel. In the end, we have build one single CAROT $\mathcal{C}_{\mathcal{A}}$ that can be used for simulating the enlarged automata $\mathcal{A}_{1/n}$ (for any $n \geq 3$) by starting with n copies of the special symbol Δ on the channel. Using ideas similar to those of Section 2.2.2 for coFlatMTL model checking, we can come with an algorithm for

model checking this timed temporal logic in the enlarged semantics, thus answering the robust model checking problem for `coFlatMTL`.

Theorem 12 ([BMR08]). *BoundedMTL and coFlatMTL robust model checking are EXPSPACE-complete.*

2.4 Conclusions and future works

My research activities on the study of timed automata is two-fold:

- in the classical framework of timed automata, my works have focused on various timed temporal logics, either for their expressiveness or for the decidability/complexity of their verification problems. The use of channel automata in the setting of timed verification is a promising technique, and may have other applications in that setting.
- in order to bring more realism in the model, and to bridge the gap between the theoretical semantics and the effective implementations of timed automata, my research has developed new techniques for *robust* model checking, where the aim is to check that a property could be preserved on a sufficiently fast hardware.

While several approaches have been proposed for the latter problem, I believe that no satisfactory solution has been reached yet: we are still missing a consensus on the *correct* semantics (if any) of timed automata. There are several directions I will explore on this topic in the coming years:

- with Patricia Bouyer-Decitre and our PhD student Ocan Sankur, we are exploring various *quantitative* approaches to robustness. This relies on bisimulation pseudometrics, which roughly measure the amount of time by which the delays must be shifted for runs of one automaton to be simulated on another automaton [HMP05, TFL10]. The value is $+\infty$ when the untimed behaviours do not match. One way of defining robustness is to require that small enlargements of guards do not take the automaton too far away for such metrics.
- in a new project between several French labs focusing on implementability and robustness, one proposal is to also consider a probabilistic view of guard enlargement. Indeed, guard enlargement considers the worst case behaviour, where all the imprecisions sum up in the most unfavourable way. While being safe, this is not very likely; moreover, since there must be a large number of (small) imprecisions in order to really make a difference, considering a probability distribution on the imprecision would probably make sense, and we could come with another quantitative approach to robustness, where we would measure how likely it is that new behaviours could appear in the implementation of a given timed automaton.
- finally, it is important to mention that the ultimate goal is to synthesise implementable controllers, in the setting of timed games (see also Section 4.3). In this

setting, the problem has more parameters: besides the imprecisions in the model of timed games (which may be handled as for timed automata), it is important to measure the implementability of the synthesized strategy: strategies where the controller has to play at very precise dates (which is actually what the classical notion of strategies assumes) may not be implementable in practice. In a recent work with Patricia Bouyer-Decitre, Marie Duflot and our master student Gabriel Renault, we proposed (in the untimed case) a notion of permissive strategies, where the aim is to synthesize (one of) the least restrictive *non-deterministic strategy*, allowing several moves when possible [BDMR09]. I would like to lift this study to the timed setting, where implementability would be measured somehow as the size of the intervals where a winning move is possible: playing at a precise date would not be implementable, while strategies allowing to play a move in a sufficiently large interval would be more realistic.

3

Quantitative verification beyond real-time

Contents

3.1	Model-checking and games on weighted timed automata . . .	58
3.1.1	Weighted temporal logics	58
3.1.2	Weighted game automata	61
3.2	Weighted timed automata under <i>energy constraints</i>	64
3.2.1	Adding constraints on the observer value	64
3.2.2	Lower-bound constraints	65
3.2.3	Interval constraints	68
3.3	Conclusions and perspectives	70

Timed automata offer a very convenient solution for reasoning about real-time systems, and additionally enjoy decidability of their reachability problem. They have received much attention from the verification community, from theoretical studies to the development of cutting-edge tools with successful applications to real case studies.

Building on this success, several other quantitative models have been defined and studied, with the aim of modeling and measuring other quantities beyond time. The very general class of hybrid automata [ACHH93, Hen96, HKPV98] extends the class of timed automata with variables satisfying more general differential equations (clock variables in timed automata satisfy the equation $\dot{x} = 1$). Unfortunately, only adding the extra differential equation $\dot{x} = 0$ (besides $\dot{x} = 1$) makes the reachability problem undecidable.

In [EMSS92], a weaker approach is proposed, considering plain, finite-state transition systems and measuring “time” by the number of transitions taken along the considered run. In this setting, the authors prove that TCTL model checking can be achieved in polynomial time. In [LMS02a, LMS06], we extended this study to weighted transition systems, where transitions carry intervals (intending to represent all *integer* values in this interval). TCTL_{≤,≥} model-checking is still in PTIME for this extension, and we proved that adding equality makes the problem Δ_2^P -complete (see Table 3 on page 77 for a definition of Δ_2^P). With Philippe Schnoebelen, we developed symbolic algorithms for these problems [MS04b], which we implemented on top of NuSMV [MS04c].

While decorating transitions with integer durations is a very weak way of representing time, it can be of interest for measuring the cost of a run, or the consumption of some resource along a run. As a consequence, it is interesting to extend timed automata with such information. Such an extension has been proposed independently by Rajeev Alur, Salvatore La Torre and George Pappas [ALP01] and by Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn and Frits Vaandrager [BFH⁺01]. The resulting model of *weighted (or priced) timed automata* just extends timed automata with *observer variables* (sometimes called *cost* variable) having piecewise-constant slopes, and which may not interfere with the behaviour of the timed automaton. Optimal reachability in these models has been proved decidable in the papers defining the models. Similar results for computing accumulated quantities along finite executions had already been obtained earlier, using closely related approaches [ACH97, KPSY99].

3.1 Model-checking and games on weighted timed automata

3.1.1 Weighted temporal logics

Building on the previous result, it is natural to extend the study of those models to richer formalisms such as temporal logics augmented with constraints on the observer variables. The most promising attempt is weighted CTL, extending the classical computation-tree logic with constraints on weights (in a similar way as TCTL extends CTL with constraints on time). Unfortunately, model checking this logic was already proved undecidable in [BBR05], with an encoding of a Minsky machine with a five-clock timed automaton with one stopwatch observer.

With Patricia Bouyer-Decitre and Thomas Brihaye, we refined this result to three-clock timed automata with one stopwatch variable: two of the clocks are used to store the values of the counters under $x_i = 2^{-c_i}$, while the third one serves as a `tick` clock. Doubling and halving are actually achieved in the same way, by building a module for testing whether the value of one clock is the double of the value of another one. This deterministic module is depicted on Figure 3.1, and runs as follows: one time unit is spent in the first two locations, so that the values of the clocks are unchanged (they are reset when they reach value 1). At the same time, the value of the stopwatch observer is augmented by the value of x_1 . The same is achieved by the middle two locations, while the last two perform the addition of $1 - y$. This way, the value of the observer variable when exiting the module has been increased by $1 + 2x_1 - y$. Hence formula

$S \wedge \mathbf{EF}_{\leq 1} T \wedge \mathbf{EF}_{\geq 1} T$ holds at the beginning of this module if, and only if, $y = 2x_1$. This provides a way of doubling (and halving) the value of x_1 : it suffices to branch the module at a point where we want to check that $y = 2x_1$, and then let y play the role of x_1 in the main part of the automaton. This way, we are able to increment and decrement the counters. A direct implementation of the whole reduction would involve four clocks, which we can reduce to three clocks with a bit of cleverness.

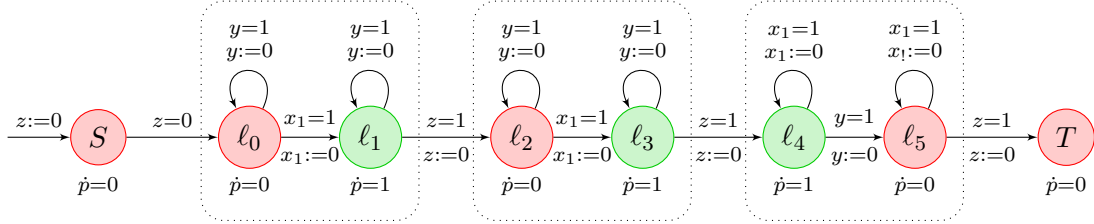


Figure 3.1: Automaton testing whether $y = 2x$

Remark. *Surprisingly, the reduction does not involve equality constraints, so that the result already holds for $WCTL_{\leq, \geq}$. The reduction can even hold for only $WCTL_{\leq}$ with minor changes (adding a path from S to a state T' along which the accumulated cost is $2 - 2x_1 + y$ and enforcing formula $S \wedge \mathbf{EF}_{\leq 1} T \wedge \mathbf{EF}_{\leq 2} T'$). This is quite surprising since optimal reachability is decidable (and the whole reduction does not involve much more than $\mathbf{EF}_{\leq c}$ constraints; however, model checking requires computing sets of configurations from which an optimal run exists).*

The above result leaves little hope for a general class of weighted timed automata with decidable model-checking problem. Still, since we did not manage to improve the above undecidability result to two-clock automata, we decided to try to prove decidability for weighted one-clock automata. One-clock timed automata have very special properties which make their analysis quite different from unrestricted timed automata [LMS04]: first, there is no need to consider integer constants other than those appearing in the automaton, which make reachability decidable in NLOGSPACE (*vs.* PSPACE for timed automata with at least three clocks). Second, resetting transitions can serve as *cycle delimiters*: contrary to general timed automata, if a resetting transition is fired twice along a run, then this defines a cycle which can be iterated.

Building on these properties, we managed to prove that model checking WCTL on one-clock timed automata is decidable, by showing that there exists an exponential-state refinement of the region automaton which is compatible with the formula being checked. This provides us with a PSPACE algorithm, labelling states with subformulas they satisfy.

Before explaining how the refinement is obtained, let us first show why it is needed. We consider the weighted one-clock automaton depicted at Figure 3.2. Thanks to a WCTL formula, we will force the cost between the a -states to be exactly 4. This will achieve the following: if the value of clock x has binary representation $x_0.x_1x_2x_3 \dots \in [0, 2)$ when leaving a , then it equals $x_1.x_2x_3 \dots \in [0, 2)$ when reaching b .

The formula we use is the following:

$$\varphi(X) = a \wedge \mathbf{E}(\neg a) \mathbf{U}_{=4} (a \wedge X).$$

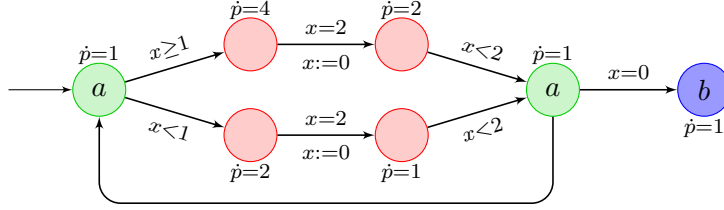


Figure 3.2: Example of a weighted one-clock timed automaton

It can then be checked that formula $\varphi(\mathbf{EX}c)$ holds in the initial state for integer values of x . By nesting φ n times, we get a formula characterizing values of the clock of the form $p/2^n$, for values of p between 0 and $2^n - 1$.

We proved that this example corresponds to the worst case, in the following sense:

Proposition 13. *Given a weighted one-clock timed automaton \mathcal{A} and a WCTL formula φ , the truth value of φ is constant over intervals of the form $(n\delta, (n_1)\delta)$, where $\delta = C^{h(\varphi)}$ with C being the l.c.m. of all weights in \mathcal{A} and $h(\varphi)$ is the constrained temporal height of the formula (i.e., maximal number of nested constrained modalities in φ).*

This result allows us to compute the truth value of φ on the refined region automaton of \mathcal{A} , in which regions have size δ . The resulting labelling algorithm, if implemented in a space-efficient manner, runs in polynomial space. In the end:

Theorem 14 ([BLM07, BLM08]). *WCTL model checking is PSPACE-complete on weighted one-clock timed automata.*

Remark. *It is worth noticing that our algorithm can cope with several observer variables, provided that each constrained modality only involves one variable. In particular, this allows us to mix timing constraints with observer constraints within the same formula.*

In the linear-time setting, we could not expect much decidability results since time can be seen as a special kind of variable with constant rate 1. Hence undecidability results for MTL directly extend to WTLTL.

The only case we managed to prove decidable is the case of weighted one-clock automata with one stopwatch observer under the timed-word semantics: we adapted the transformation of MTL formulas to alternating one-clock timed automata, where the clock is actually a stopwatch. The arguments follow those for proving decidability of MTL over finite timed words, by abstracting the product of the automaton under checking and the alternating timed automaton. That this abstraction is time-abstract bisimilar with the original product crucially relies on the restriction to a one-clock automaton and

a stopwatch observer. The rest of the proof uses classical well-quasi-order arguments, as explained in Section 2.2.1.

We proved that any natural extension to the model leads to undecidability [BM07]:

- one-clock automata with one (non-stopwatch) observer;
- one-clock automata with two stopwatch observers;
- two-clock automata with one stopwatch observer.

These reductions are again from Minsky machines, with the values of both counters being encoded in the single clock with $x = 2^{-c_1} \cdot 3^{-c_2}$. The module for halving the value of the clock is depicted on Figure 3.3, where we impose that the part of the run from a to b has cost 1 (and no time elapses in a and b). It can be checked that this divides the value of x by 2. The case of two-clock automata with one stopwatch observer is achieved by duplicating states with observer-rate k with k states with rate 1 (the time spent in the state with rate k being stored in the second clock). Finally, the case of two stopwatches is handled by adapting the two-clock case, simulating one of the clocks with an observer variable, and transferring clock constraints of the automaton into the formula.

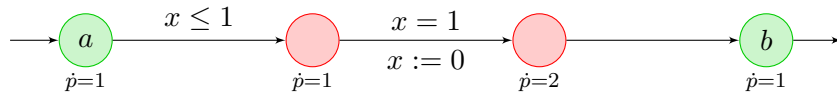


Figure 3.3: Module for halving the value of the clock

3.1.2 Weighted game automata

The extension with observer variables is especially interesting in the setting of games, as it allows to model hybrid variables in real-time open systems. We assume here two-player timed games, where the set of transitions is partitioned into *controllable* and *uncontrollable* ones. Our decidability results however only apply to *turn-based* timed games.

Unfortunately again, there is little hope for getting a general decidability result as was obtained for optimal reachability: the undecidability result we obtained for WCTL immediately extends to optimal reachability in the game setting. Indeed, we used the branching capabilities of WCTL to check certain properties along a main path which encodes a run of a two-counter machine. In a two-player setting, the antagonist has the ability of running these checks all along the computation.

Theorem 15 ([BBM06]). *Optimal reachability is undecidable in two-player weighted timed games with at least three clocks.*

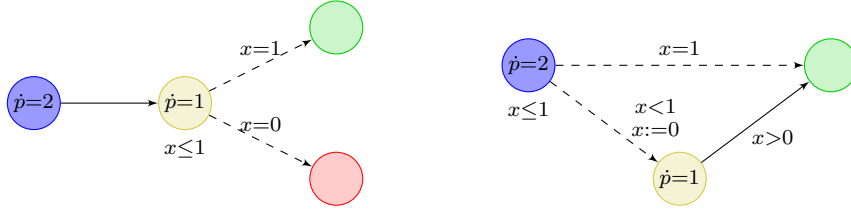


Figure 3.4: Examples of weighted timed games

Looking again on the one-clock side, we managed to obtain some positive results. The idea is to inductively compute the smallest cost the protagonist can achieve for reaching a target location (in the worst case, *i.e.*, whatever the antagonist does).

Let us first mention that such optimal strategy need not exist: the weighted timed game on the left of Figure 3.4 displays an example where this is not the case (where dashed transitions are uncontrollable, and where the aim is to reach the top-right position): the protagonist has to spend a positive delay in the first location, in order to prevent the antagonist to send the game to the bottom (losing) state. But any positive delay can be shortened, always yielding a better strategy. In the same way, when optimal strategies exist, they may require memory, as exemplified by the weighted timed automaton on the right of Figure 3.4: if the play ever goes to the bottom state, when the protagonist can, by playing quickly enough, enforce total cost less than 2. However, how quick is *quickly enough* depends on the delay spent in the initial state; this information is lost upon resetting the clock. Notice that both examples are actually turn-based.

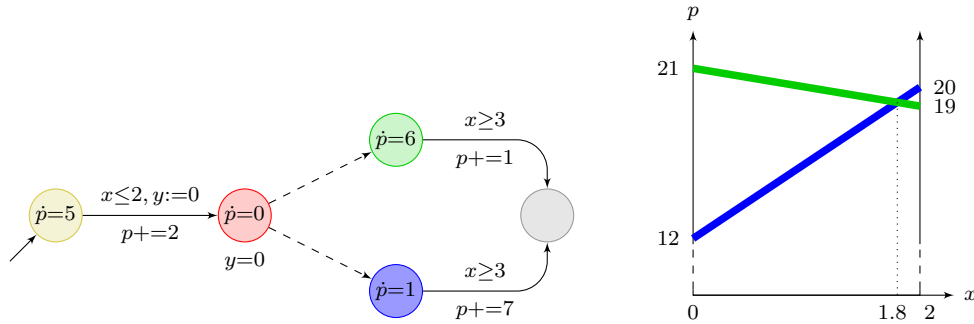


Figure 3.5: Example of a weighted timed game

What we proved is that there always exist *memoryless* ε -optimal strategies for weighted one-clock *turn-based* timed games. One easily gets convinced that those strategies cannot be region-based: Figure 3.5 displays an example of such a game, where the optimal time to leave the first location is 1.8. This can be checked by computing the cost of a run spending x time unit in the initial location, and visiting the bottom state (resp. the top state), as depicted on the right of Figure 3.5.

We now explain the main lines of our algorithm. For a location ℓ and a clock value x , we write $\text{cost}(\ell, x)$ for the optimal cost that can be secured by the protagonist. Our algorithm computes this value for all reachable configurations (ℓ, x) of the game.

We extend the model by allowing special locations labelled with functions $x \mapsto \text{cost}(x)$, which indicates the optimal cost of reaching the target in this state depending on the value of the clock. Once the information has been computed for a location, we replace this location with such a cost function, and proceed to other locations. There are two important arguments for ensuring termination of this procedure:

- as a preliminary step, we unwind the game in such a way that the same resetting transition will not be taken twice. Indeed, taking such a transition always leads to the same configuration, which means that it is either losing or not optimal. We prevent this by making the second occurrence of each resetting transition losing.
- the inductive step consists in considering the location with minimal cost rate in the automaton:
 - if it is controlled by the protagonist, then we prove that there is no interest in delaying twice in this location (for a reason similar to the one above). We can thus (inductively) compute the optimal cost in the game without this location, then compute the optimal cost from this location, and finally replace this location with the computed cost function.
 - if it is uncontrollable, there is intuitively no incentive for the antagonist to delay in this location (since it has the smallest of all rates), unless we are “close” to the goal. Then we can mark this location as urgent (*i.e.*, we prevent time to elapse in this location), set its cost rate to $+\infty$ (as no time will elapse anyway), and allow an extra outside cost function (corresponding to the case where the antagonist would still want to delay in this location, because the goal will be reached very soon).

Applying this procedure inductively, we eventually end up with a game where all locations are urgent, which is then untimed and can be solved easily.

In the end, what we prove is that the cost function in each location is piecewise affine, with slopes corresponding to cost rates of locations of the automaton. We get:

Theorem 16 ([BLMR06]). *Optimal reachability is decidable in turn-based weighted one-clock timed games.*

Our procedure runs in 3-EXPTIME, as we have to unwind the game several times. A more careful implementation can however improve the running time of the algorithm to a single exponential [Rut11]. To the best of our knowledge, the best lower-bound is PTIME, which still leaves room for further improvement.

Remark. In [BCFL04], a pseudo-algorithm has been proposed for computing optimal reachability cost in turn-based weighted timed games (no restriction on the number of clocks). The procedure just computes over-approximations of the optimal cost in a backward manner. Termination was proved only under the assumption that the cost increases by at least a (fixed) amount along any cycle of the region automaton. From our results above, we also get termination of the procedure under the one-clock restriction.

3.2 Weighted timed automata under *energy constraints*

3.2.1 Adding constraints on the observer value

Weighted automata and games have proven very convenient for modelling and measuring resource consumption (energy, bandwidth, ...) in embedded systems. In the case of energy, however, the above solution is not always adequate for autonomous systems: such systems may not only consume, but also regain energy. In that setting, the main aim is not optimization, but rather management of the resource, in such a way that the resource is never exhausted.

During the last three years, I've been involved in the European project Quasimodo, focusing on quantitative approaches in model-driven development. This project involves several industrial partners, among which Hydac (a company producing hydraulic accumulators, pumps, ...) proposed a similar problem: they have a machine pumping oil from a tank, at certain rates depending on time. They also have a pump for filling in the tank. They want the tank to never be empty, and to never overflow. As an additional requirement, they would also like to minimize the level of oil in the tank, since high levels of oil lead to high pressure in the tank, which may damage the pumps.

Weighted timed automata are the model of choice for such problems. However, while optimal reachability could cope with negative rates [BBBR07], we never imposed any safety conditions to be satisfied along the runs. We thus defined a new semantics for weighted timed automata, in order to allow such constraints. Given a lower bound L , the L -semantics of a weighted timed automaton is an infinite-state system where each state records the valuation of the clocks and the value of the observer, with the natural transitions between states, and with the following amendment: from states where the observer has value below L , there is no outgoing transition. If we are additionally given an upper bound U , then we also drop transitions out of states where the observer value exceeds U . A run in a weighted timed automaton is said *feasible* if it satisfies the associated energy constraints.

It must be noticed that this new semantics is a slight departure from the motto that observer variables would not interfere with the semantics, hence it goes one step further towards hybrid automata. Still, we managed to obtain decidability results (mostly in the untimed and one-clock cases), which we present below.

A related problem has been studied in the setting of *interfaces* [dAH01]: roughly, interfaces express constraints on the input that is fed to a component in order to ensure some correctness properties on its output. This provides a way of deciding whether two components are *compatible*. The theory of interfaces has been extended with constraints on resources, for modelling consumption or production of some resource. Compatibility then includes a quantitative notion for modelling production and consumption of some resource: in order to be compatible, the first component must produce more resources than what the second component consumes. Deciding compatibility between resource interfaces is shown decidable in polynomial time [CdAHS03].

3.2.2 Lower-bound constraints

In the case where only lower-bound constraints are imposed on the energy variable, optimization still makes sense: if a run with a given initial credit is feasible, then the same run can be mimicked with any higher initial credit. The aim then remains to maximize the value of the observer variable (contrary to the previous section, where we aimed at minimization), with the additional constraints that it may not drop below a certain value.

Following this remark, we adapted the Bellman-Ford algorithm to handle the untimed case: our algorithm iteratively computes the maximal value that can be accumulated in at most k steps from the initial state and with a given initial credit, under energy constraints. In case this still increases after $2 \cdot |S|$ steps, then the value is unbounded. Otherwise, the algorithm computes the optimal accumulated cost that can be achieved from the initial state to each state of the automaton. Hence:

Theorem 17 ([BFL⁺08]). *Optimal reachability under lower-bound constraint is decidable in polynomial time on (untimed) weighted automata.*

In the case of (safety) games, we proved very tight links between energy constraints and the classical *mean-payoff* constraint [EM79, ZP96]:

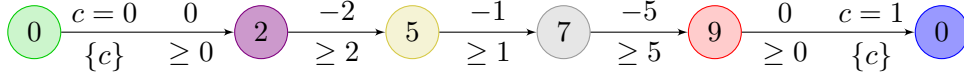
Proposition 18. *Energy games admit memoryless optimal strategies for both players, and are logspace-equivalent to mean-payoff games.*

Remark. *It must be noticed that energy constraints have been much studied since we introduced them in [BFL⁺08], especially in the untimed case. In particular, extending a previous work by Yuri Lifshitz and Dmitri Pavlov [LP07], Luboš Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini and Jean-François Raskin used energy games to improve the best-known algorithm for solving mean-payoff games [DGR09, BC10]. Energy-constraint conditions are now also considered as one of the important quantitative objectives in games, as is witnessed by the impressive amount of work they have received over the last year [BC10, CD10, CDHR10, DDG⁺10].*

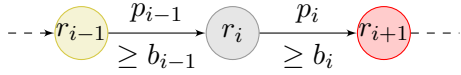
In the timed setting, the situation becomes much harder. Our current studies have only focused on one-clock automata, and are, to the best of our knowledge, the only (published) results available in the timed case.

Taking advantage of the restriction to one-clock automata, we first considered the problem of optimizing the final value of the observer along a single, non-resetting timed path, depending on the initial credit. This restriction already contains much of the difficulty of the problem, as we explain later.

A *unit path* is a non-branching acyclic timed automaton whose traversal takes exactly one time unit. Transitions may carry integer values, which will be added to the observer value upon taking the transition. We also allow transitions to carry lower-bound constraints on the value of the observer. As a running example, we consider the following unit path:



Notice that in this example, the cost constraints on transitions exactly match the discrete costs: this simply ensures that the value of the observer will remain nonnegative all along the run.



We proved that the following intuition is correct along such paths: at any time, it is always more profitable to delay in the reachable location with highest rate. This entails that an optimal policy would spend

no time in the i -th location if one of the following two conditions hold:

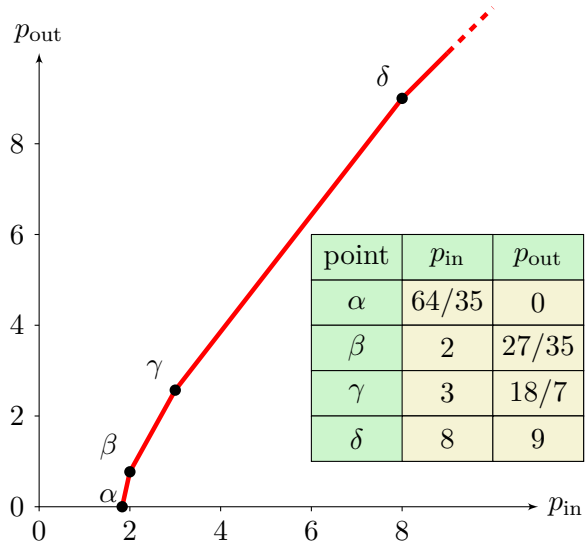
$$r_{i-1} > r_i \quad \text{or} \quad r_{i+1} > r_i \text{ and } b_{i-1} + p_{i-1} \geq b_i \quad (3.1)$$

When one of the above two conditions is fulfilled, we can drop location r_i , replacing it with a transition from r_{i-1} to r_{i+1} with cost constraint $\geq \max\{b_{i-1}, b_i - p_{i-1}\}$ and discrete cost $p_{i-1} + p_i$. A path where no state satisfies (3.1) is then said to be in normal form. Along a path in normal form, we have:

Proposition 19. *The optimal observer value with which to exit the i -th location is b_i .*

This property is proven with very basic techniques, showing that a run not following this policy can be improved. Of course, the overall timing constraint has to be taken into account: if it is not possible to match a cost constraints within one time unit, this simply means that the initial value was not high enough.

For our example, our algorithm finally computes the curve on the right, representing the optimal exit value in terms of the initial value. For instance, if we enter the path with initial observer value 2, the optimal policy is to spend no time in the location with rate 2 (as we can leave it directly for a more interesting location), then spend $1/5$ time units in the next location (so that we have value 1 and can fire the outgoing transition), then spend $5/7$ time units with rate 7, and the remaining $3/35$ time units in the location with rate 9, ending with final observer value $27/35$ (point β). This computation also indicates that there



is no way of reaching the end of the path if starting with credit lower than $64/35$. Finally,

notice that point δ corresponds to the case where the initial credit allows to directly rush to the most profitable location, and spend one time unit there. When the initial credit is higher, then the extra amount is “unused”, and remains available at the end, so that the slope after point δ is 1.

Interestingly, we could carry a similar study for an *exponential observer variable*: for such an observer, a location with rate k indicates that the observer value satisfies the differential equation $\dot{p} = k \cdot p$. As a consequence, the observer value evolves as an exponential function of time, hence its name. Notice that discrete updates on transitions remain additive, so that this case is not simply obtained by “exponentiating” the previous case.

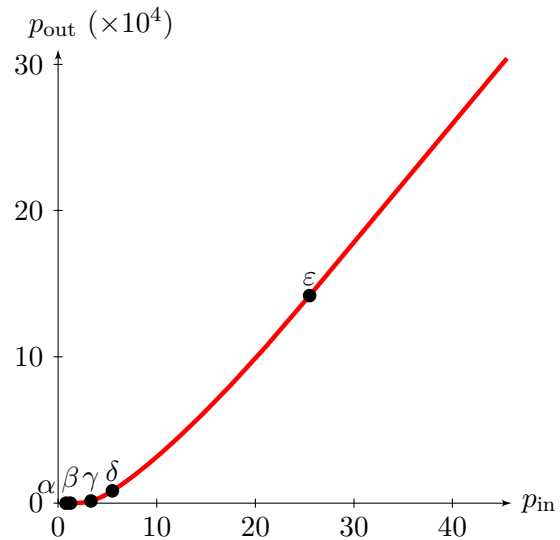
For paths such as the one of our running example, with positive rates and negative updates and with zero as the global lower-bound cost constraint, we could also compute the optimal schedule to be applied for maximizing the final observer value, depending on the initial credit. This also uses a normal-form transformation as in the linear case: this time, the condition for spending no time in a location (hence for dropping this location) writes as follows:

$$\frac{p_{i-1} \cdot r_{i-1} \cdot r_i}{r_{i-1} - r_i} > \frac{p_i \cdot r_i \cdot r_{i+1}}{r_i - r_{i+1}} \quad (3.2)$$

Along a path in normal form with exponential observers, we have:

Proposition 20. *The optimal observer value with which to exit the i -th location is $(p_i \cdot r_{i+1}) / (r_{i+1} - r_i)$.*

In the end, we could also compute (in closed form) the optimal delays for exponential observers. For our running example, we obtain the curve depicted on the right, where for instance the minimum initial credit to be able to reach the final location is $\exp(-2) \cdot 10/3 \cdot (21/8)^{2/5} \cdot 2^{2/7}$, which is around 0.81 (corresponding to point α on the curve). We proved also that the curve, while being piecewise of the form $p_{\text{out}} = M \cdot (p_{\text{in}} - q)^{r_i/r_j}$, has continuous derivative at junction points (where r_i and r_j are cost rates from the automaton).



Lifting this study on paths to general one-clock timed automata is achieved by considering all the locations of the automaton which can be entered by a resetting transitions. All the paths between two such locations can be handled using the above procedure on paths, so that we end up with a finite-state automaton in which transitions are labelled with the

observer transformation they achieve. These functions have particular properties (*e.g.*, their derivatives, when defined, are always larger than or equal to 1), from which we get that their set of fixpoints, if non-empty, is an interval, and only for the points in or on the right of this interval can the function be iterated infinitely many times. Proving that we can restrict to lasso-shaped runs completes the proof, yielding the following result:

Theorem 21 ([BFLM10]). *Optimal reachability is decidable in exponential time in one-clock weighted timed automata under lower-bound constraint.*

Remark. *In the case where all discrete observer updates on transitions are zero, the algorithm is (conceptually) much simpler: it suffices to directly jump to the most profitable location and spend the available time there. This can be handled using the corner-point abstraction, which we already mentioned at Section 1.3.3. In the one-clock case, this provides us with a polynomial-time algorithm [BFL⁺08].*

Finally, notice also that discrete costs can be modelled using a second clock, so that this simple argument does not extend to the n -clock setting.

3.2.3 Interval constraints

Interval constraints in the untimed case are quite easy to handle: since the number of allowed values for the observer variable is bounded (by the size of the interval), we can include this value explicitly in the automaton. This involves an exponential blowup, which we proved is unavoidable for games (EXPTIME-complete). In the case of reachability under interval constraints, an on-the-fly implementation requires polynomial space, while we could only prove an NP lower-bound¹ (by a direct encoding of the SUBSET-SUM problem).

Theorem 22 ([BFL⁺08]). *The existence of a safe strategy under interval constraints in (untimed) weighted automata is decidable in polynomial space, and is NP-hard.*

In the timed case, the above argument obviously fails. Actually, we proved that reachability is undecidable already in the one-clock case, encoding the halting problem for two-counter machines. This time, the values of the counters are encoded in the value of the observer, with the following correspondence: writing c_1 and c_2 for the two counters, the observer variable has value $5 - 2^{-c_1} \cdot 3^{-c_2}$ when clock $x = 0$. The weighted timed game depicted on Figure 3.6 achieves incrementation and decrementation of the counters (depending on the value of n): starting with initial value $5 - e$, the observer has value $5 - \frac{ne}{6}$ when leaving this module, unless state `ok` is reached. We may only be in m_1 with value 0 for the variable, since otherwise the antagonist may force the game spend one time unit in the state with rate 5, hence exceeding the upper bound. Taking for instance $n = 3$ achieves incrementation of counter c_1 , while $n = 18$ decrements counter c_2 .

¹This problem reduces to the reachability problem for two-clock timed automata, whose exact complexity is also open, between NP and PSPACE.

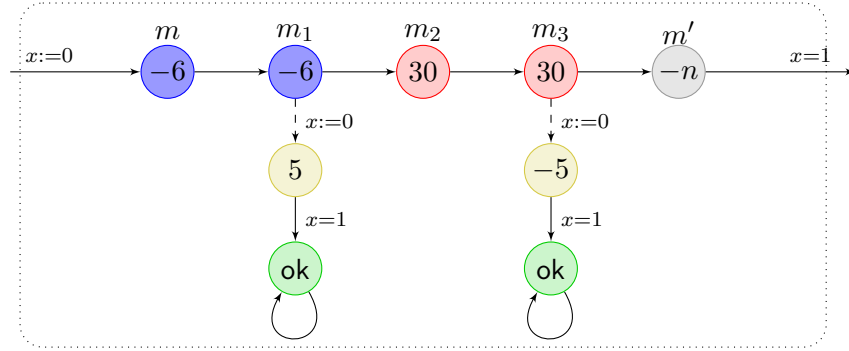


Figure 3.6: Generic module Mod_n

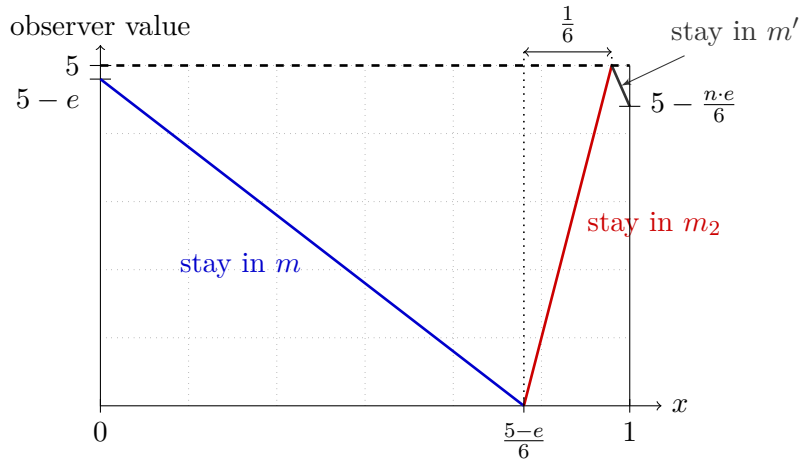


Figure 3.7: The effect of module Mod_n

Using this game, we can easily check whether the values of the counters are zero or not. Conditional commands of the form

$$\text{if } c_1 = 0 \text{ then goto } p_1 \text{ else goto } p_2$$

can then be handled as in the game depicted on Figure 3.8: the protagonist chooses whether she wants to go to p_1 or p_2 , and the antagonist has a way to check if this choice agrees with the value of counter c_1 .

Theorem 23 ([BFL⁺08]). *The existence of a safe strategy under interval constraints is undecidable in one-clock weighted timed automata.*

Notice that this example does not use discrete costs on transitions. Using discrete costs, we can get rid of the second player in the module of Figure 3.6, by inserting two consecutive transitions with costs $+5$ and -5 : such a sequence of transitions can only be taken if the value of the variable is 0. In a very recent work with Patricia Bouyer-Decitre, we managed to adapt the above undecidability proof to the two-clock reachability problem,

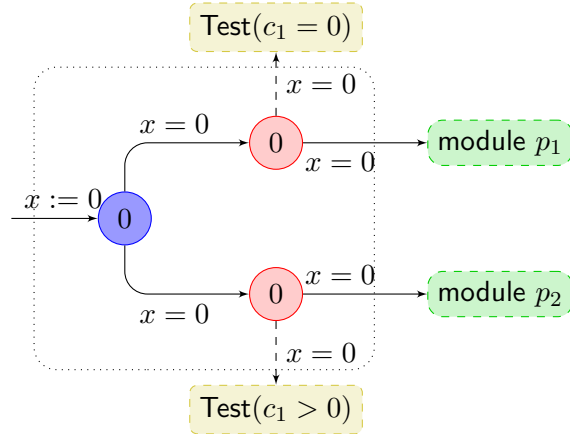


Figure 3.8: Module for conditional commands

getting rid of the second player in the game of Figure 3.8 by storing the information on e in the extra clock before performing the test.

Claim 24 (Unpublished). *Reachability under interval constraints is undecidable in two-clock weighted timed automata.*

In a recent (unpublished) work with Pierre-Alain Reynier, presented at the 2010 workshop of the GASICS project, we have studied this problem in a setting where the cost rates are not known exactly: locations are not labelled with one single rate, but with a (non-punctual) interval of possible rates. The exact rate is uncontrollable, and the value of the variable is only observed along some transitions (called *observation transitions*). Our preliminary results are very promising:

Theorem 25 (Unpublished). *Under some assumptions (observation transitions reset the clock, at least one observation transition along any cycle, strong non-Zenoness), reachability is decidable in one-clock weighted timed games with imprecisions under interval-constraints.*

This direction of research is especially interesting since we trade non-robustness for decidability.

3.3 Conclusions and perspectives

Weighted timed automata are a very interesting extension of timed automata, especially well-suited for modelling resource consumption: they extend timed automata with *observer variables*, which can be used to measure other quantities beyond time. While optimal reachability is decidable on these models, we have to impose severe restrictions on the model (namely, restrict to the one-clock setting) to ensure satisfiability of checking stronger requirements.

Energy constraints are a particularly interesting semantics for weighted timed automata, where the aim is not optimization but rather *management* of resources: in that setting, we are after a schedule that allows some level (battery charge, oil in a tank, ...) to stay within given bounds. Again, in restricted cases, we obtained interesting decidability results for this kind of problems.

Our 2008 paper on energy constraints have raised an enormous interest for that problem. Still, many problems remain open, and we are trying hard to solve some of them. In particular, there is still hope that the lower-bound problems (both w.r.t. optimal reachability and optimal strategy) would be decidable even if we drop the one-clock restriction.

While we only worked here on weighted timed automata with one observer variable, the extension to multiple variables could also turn out to be interesting. As already mentioned, our WCTL model-checking algorithm for the one-clock setting could cope with several variables, provided that only one constraint is imposed at a time. Also, optimization of one variable under constraints on the other variable(s) has also been proved decidable in [LR08] (assuming *non-decreasing* variables). When turning to energy constraints, the situation is less appealing, as two variables with lower-bound constraints can encode one variable with interval-bound constraints, where most problems are undecidable.

There are several extensions of this work I would like to focus on a near future. The first one is mostly theoretic: weighted timed automata under energy constraints can be seen as an extension to a continuous setting of one-counter finite-state automata. These automata are restrictions (to a one-letter alphabet) of pushdown automata. It then becomes natural to consider timed automata with *continuous pushdown*: the timed automaton would be able to store a real-valued amount of several *products*, but could only consume the top-most product on the stack, with the aim of never running out of the product it has to consume in a given state.

A second, more practical problem is the extension of our preliminary results mentioned in Theorem 25: assuming imprecision in the values of the clocks and/or in the values of the cost rates, most of the undecidability proofs for weighted timed automata would fail, as they rely on a very precise encoding of two-counter (or Turing) machines. I think this is a very promising direction, in some sense similar to the decidability result obtained for lossy-channel automata [AJ96] as opposed to the undecidability of classical channel automata.

4

Verification of concurrent games

Contents

4.1 Alternating-time temporal logic	73
4.1.1 Expressiveness issues	74
4.1.2 ATL model checking	75
4.2 Extending ATL with strategy contexts	78
4.2.1 Expressiveness of ATL with strategy contexts	79
4.2.2 Model checking algorithm for ATL* with strategy contexts	81
4.2.3 A note on strategy logic	85
4.3 Really concurrent timed game automata	85
4.3.1 Timed concurrent game structures	86
4.3.2 Properties of strategies in timed CGSs	87
4.3.3 Timed ATL model checking on timed CGSs	88
4.3.4 Ruling out Zeno strategies	89
4.3.5 Extensions of the model	90
4.4 Conclusions and future works	90

4.1 Alternating-time temporal logic

ATL* was defined in [AHK97] as an extension of CTL* for games. Using *strategy quantifiers* instead of classical *path quantifiers*, it provides a way of expressing *controllability properties*, which are of interest in the study of open systems.

ATL* (and its fragment ATL) has been quite extensively studied over the last ten years: ATL model checking has been settled PTIME-complete while its satisfiability problem is EXPTIME-complete; ATL* model checking and satisfiability problems are 2-EXPTIME-complete.

In this chapter, we begin with having a closer look at the classical setting of ATL, showing some new results on expressiveness and model-checking complexity. We then propose an extension of ATL towards expressing non-zero sum properties, in which the players may have to take advantage of the other players' strategies in order to achieve their goals. Finally, in the third section, we extend concurrent game structures with time, obtaining a new kind of timed games which we prove are very expressive while still enjoying a decidable model-checking problem.

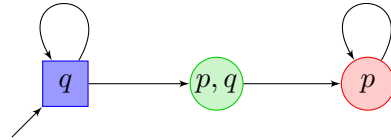
4.1.1 Expressiveness issues

I started working on temporal logics for games in 2005, with François Laroussinie and Ghassan Oreiby. Our aim was to extend ATL to the timed setting. We begun with having a close look at the original papers on ATL [AHK97, AHK02]. In these papers, following Equation (1.1) stating that CTL can be defined using only modalities **EX**, **EG** and **EU**, the syntax of ATL is given as

$$\text{ATL} \ni \varphi_s ::= p \mid \neg \varphi_s \mid \varphi_s \vee \varphi_s \mid \langle\langle A \rangle\rangle \mathbf{X} \varphi_s \mid \langle\langle A \rangle\rangle \mathbf{G} \varphi_s \mid \langle\langle A \rangle\rangle \varphi_s \mathbf{U} \varphi_s \quad (4.1)$$

Several authors have used this syntax in their subsequent works on ATL [vD03, KP04, HP06, GvdD06, WvdHW07, JD08, ALNR10, among others].

However, as witnessed by the (turn-based) game automaton depicted on the right, Equation (1.1) cannot be lifted to ATL: the existence of a strategy (for the circle-player) enforcing $p \mathbf{R} q^1$ does not imply the existence of a strategy enforcing $\mathbf{G} q$, nor does it imply the existence of a strategy enforcing $q \mathbf{U} (p \wedge q)$.



Actually, we proved that this version of ATL cannot express $\langle\langle A \rangle\rangle p \mathbf{R} q$. Following Table 1, the proof consists in building two infinite families of game automata, one of which satisfies $\langle\langle A \rangle\rangle p \mathbf{R} q$, and prove that the i -th game automaton of each family cannot be distinguished by ATL_r formulas of size less than i . Both families are depicted (as one game automaton with several possible initial states) on Figure 4.1. It can be checked that the difference between states s_i and s'_i is the existence of a fourth move allowed to Player 1 in s'_i . It is easily checked that playing move 4 in s'_i is a winning strategy for Player 1 for the objective $q \mathbf{R} (p \vee q)$ (which is equivalent $\mathbf{G} p \vee p \mathbf{U} q$). Notice that an outcome in \mathcal{C}_i satisfies $q \mathbf{R} (p \vee q)$ if, and only if, it does not visit s_0 . It is easily observed that there is no strategy for Player 1 to surely avoid reaching s_0 .

On the other hand, by an inductive proof, it can be shown that no ATL-formula (as defined at Equation (4.1)) of size less than i can distinguish between s_i and s'_i . Hence:

¹ \mathbf{R} is the *release* modality, which is dual to *until*. Its semantics is given by $p \mathbf{R} q \equiv \neg((\neg p) \mathbf{U} (\neg q))$. Equivalently, from Equation 1.1, we have $p \mathbf{R} q \equiv \mathbf{G} q \vee q \mathbf{U} (p \wedge q)$.

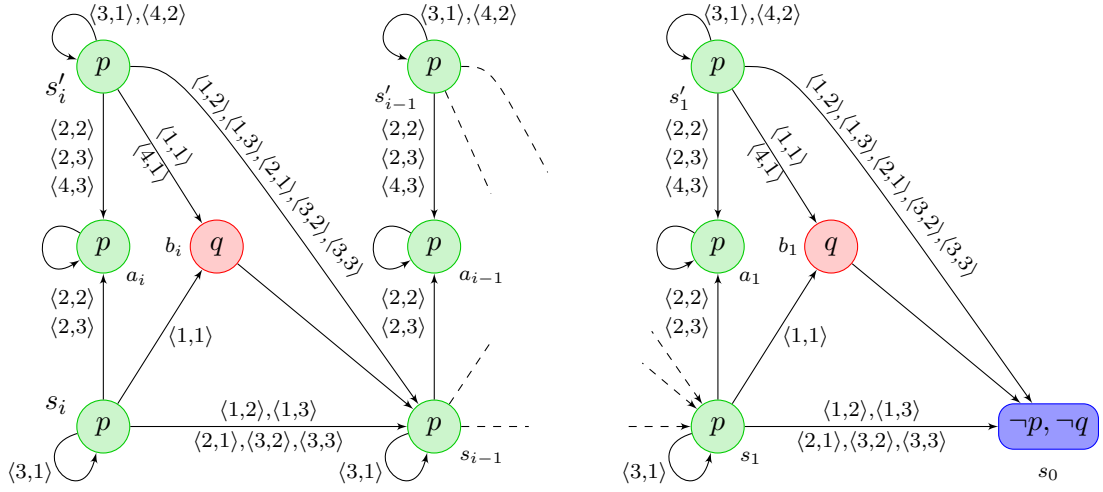


Figure 4.1: The game automaton \mathcal{C}_i , with states s_i and s'_i on the left

Theorem 26 ([LMO07]). *ATL (as defined at Equation (4.1)) cannot express $\langle\langle A \rangle\rangle p \mathbf{R} q$.*

Remark. *It must be noticed that the above results fails over turn-based games: indeed, since turn-based games are determined [Mar75], formula $\langle\langle A \rangle\rangle p \mathbf{R} q$ is the exact opposite of $\langle\langle \text{Agt} \setminus A \rangle\rangle (\neg p) \mathbf{U} (\neg q)$, which is an ATL formula with the above definition.*

Notice also that the results proved in aforementioned papers involving the weaker definition of ATL extend to the more expressive version.

On a similar note, it is well-known that CTL cannot express \mathbf{EF}^∞ [EH86], but $\mathbf{AF}^\infty \varphi \equiv \mathbf{AG} \mathbf{AF} \varphi$. Dually, \mathbf{EG}^∞ can be expressed in CTL while \mathbf{AG}^∞ cannot. The situation is different with ATL, since $\langle\langle A \rangle\rangle$ generalizes both path quantifiers.

Proposition 27 ([LMO07]). *Neither $\langle\langle A \rangle\rangle \mathbf{F}^\infty$ nor $\langle\langle A \rangle\rangle \mathbf{G}^\infty$ can be expressed in ATL.*

4.1.2 ATL model checking

ATL model checking was studied in the papers introducing the logic [AHK97, AHK98, AHK02], on different families of n -player game automata. The most general model was that of CGSs, for which ATL model checking was proved PTIME-complete [AHK02].

This is proved by a labelling algorithm similar to that for CTL model checking. Each modality is handled by a fixpoint computation, consisting in computing *controllable predecessors*, instead of simple *predecessors* for CTL (see Section 1.2.2).

However, this result assumes that the transition table is given explicitly in the CGSs, which is not convenient: the transition table has size exponential in the number of players, as it associates one transition with each combination of possible moves of each player. To overcome this problem, we introduced *symbolic CGSs*, in which the transition table is encoded as boolean formulas built on the following grammar:

$$\text{MConstr}(\text{Agt}, \text{Mov}) \ni \varphi ::= \top \mid A = m \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg \varphi$$

where A ranges over Agt and m over Mov . The intended meaning of $A = m$ is “Player A plays move m ”.

Definition 28 ([LMO07]). A symbolic CGS is a 5-tuple $\mathcal{G} = \langle \mathcal{S}, \text{Agt}, \text{Mov}, \text{Chc}, \text{Edg} \rangle$ where \mathcal{S} is the underlying transition system, Agt is a finite set of agents who select actions from the finite set Mov . For each state $s \in \mathcal{S}$, $\text{Edg}(s)$ is a finite sequence $(\varphi_i, t_i)_{i \in I}$ where for all $i \in I$, φ_i is a formula in $\text{MConstr}(\text{Agt}, \text{Mov})$ and t_i is a transition in T .

For instance, “from q_1 , if Player A_1 plays move m_2 or m_3 and Player A_3 plays m_2 , then go to state q_5 , otherwise go to q_2 ” would be encoded as

$$\text{Edg}(q_1) = \{([A_1 = m_2 \vee A_1 = m_3] \wedge A_3 = m_2, (q_1, q_5)), (\top, (q_1, q_2))\}.$$

In our setting of *deterministic games*, we take the transition given by the first constraint being satisfied. However, this could also be used to encode non-deterministic games, where several transition can be associated to a move vector.

Obviously, CGSs and symbolic CGSs have the same expressive power, but the latter can in some cases be more succinct. This, of course, has a price: model-checking cannot be achieved in polynomial time anymore. Actually, computing controllable predecessors becomes much more expensive (see Table 3 for definitions of those complexity classes):

Proposition 29. Computing $CPre$ is Σ_2^P -complete on symbolic CGSs, while it is in AC^0 on classical CGSs.

This increases the *program complexity* of model checking for ATL and its extensions. Concerning the overall complexity, we get:

Theorem 30 ([LMO07, LMO08]). On symbolic CGSs, ATL and EATL model checking is Δ_3^P -complete, ATL^+ model checking is PSPACE-complete², and ATL^* model checking is 2-EXPTIME-complete.

The polynomial-time hierarchy [Sto76]. Given a language \mathcal{L} (think for instance of the set of words representing all positive instances of the NP-complete problem SAT), a *Turing machine with oracle \mathcal{L}* is a Turing machine equipped with an extra tape (the *oracle tape*) and three special states q_{oracle} , q_{yes} and q_{no} . During the computation, whenever the Turing machine enters state q_{oracle} , it directly jumps to q_{yes} if the word on the oracle tape is in \mathcal{L} , and to q_{no} otherwise.

We write $\text{PTIME}^{\mathcal{L}}$ for the set of languages accepted by deterministic Turing machines with oracle \mathcal{L} and halting in polynomial time (in the size of the input). We write $\text{NP}^{\mathcal{L}}$ for the analogous class defined with non-deterministic machines. When \mathcal{C} is a class of languages (in particular, when it is a complexity class), we write $\text{PTIME}^{\mathcal{C}}$ for the union of all languages in $\text{PTIME}^{\mathcal{L}}$ when \mathcal{L} ranges over \mathcal{C} . $\text{NP}^{\mathcal{C}}$ is defined analogously.

The polynomial-time hierarchy is the following sequence of complexity classes, defined recursively with $\Sigma_0^{\text{P}} = \Pi_0^{\text{P}} = \Delta_0^{\text{P}} = \text{PTIME}$ and

$$\Sigma_{i+1}^{\text{P}} = \text{NP}^{\Sigma_i^{\text{P}}} \quad \Pi_{i+1}^{\text{P}} = \text{coNP}^{\Sigma_i^{\text{P}}} \quad \Delta_{i+1}^{\text{P}} = \text{PTIME}^{\Sigma_i^{\text{P}}}$$

The classes in this hierarchy lie between PTIME and PSPACE . A natural problem in Σ_i^{P} is the problem QSAT_i of finding the truth value of

$$\exists X_1. \forall X_2. \exists X_3 \dots Q_i X_i. \varphi(X_1, X_2, X_3, \dots, X_i)$$

where φ is a boolean formula with variables in X_1 to X_i . The dual problem (where the sequence of quantifications begins with a universal one) is Π_i^{P} -complete. The problem SNSAT_i , made of several instances of QSAT_i where the k -th instance uses the truth value of the $k-1$ previous ones, is an example of a Σ_{i+1}^{P} -complete problem.

Circuit complexity classes [Vol99]. A Boolean circuit is a finite directed acyclic graph having n input gates (nodes with no predecessors) and m output gates (nodes with no successors). All gates (except input gates) are labelled with a boolean function (such as conjunction, negation, ...) Such a circuit associates, in a natural way, an m -bit output value to an n -bit input value.

A language \mathcal{L} in $\{0, 1\}^*$ is in a given circuit complexity class if for all n , one can build a circuit with n input gates, one output gate, polynomially many gates, polylogarithmic depth, such that a word of size n is in \mathcal{L} iff the circuit above associates 1 to this input word. There are three important parameters in this definition:

- the depth of the circuit: by definition, it is in $O(\log^i(n))$, where $i \in \mathbb{N}$. The case where $i = 0$ corresponds to constant-depth circuits;
- the set of allowed gates: negation, conjunction and disjunction with bounded fan-in define the classes NC^i . Adding unbounded-fan-in conjunction and disjunction gives the classes AC^i . Finally, adding the *majority* gate (returning 1 if at least half of its inputs is 1) defines TC^i .
- the type of Turing machine used to build the circuit, given the length n of the input. It is standard to allow logarithmic space (when n is given in unary).

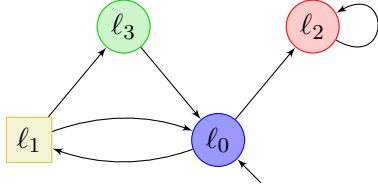
These classes satisfy the following inclusions:

$$\text{NC}^i \subseteq \text{AC}^i \subseteq \text{TC}^i \subseteq \text{NC}^{i+1} \subseteq \text{PTIME.} \quad \text{and} \quad \text{NC}^1 \subseteq \text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{AC}^1$$

A natural problem complete for NC^1 is the evaluation of a boolean formula [Bus87].

Table 3: Complexity classes above and below PTIME

4.2 Extending ATL with strategy contexts



The semantics of ATL is, in some cases, not the one one can expect. Consider the (two-player turn-based) game on the left, between Player A_1 , who controls circle states, and Player A_2 , who controls the square state. Consider the following ATL formula:

$$\langle\langle A_1 \rangle\rangle \mathbf{G} (\langle\langle A_2 \rangle\rangle \mathbf{F} l_3). \quad (4.2)$$

This formula can be understood as follows: “Player A_1 has a strategy under which it will always be possible for A_2 to reach l_3 ”. One would expect this statement to be true, if the circle player plays to the left in the initial state and the square player moves to l_3 .

In ATL, Formula (4.2) actually means that “Player A_1 has a strategy to only visit states from which A_2 has a strategy to reach the green state”. However, already in the initial state, Player A_2 does not have such a strategy.

I started working on this topic with François Laroussinie in 2007, and we hired Arnaud Da Costa Lopes as a PhD student for working on the subject. At the very same time, several related works were published (listed chronologically; all four papers were published between June and October 2007):

- ATL with irrevocable strategies [ÅGJ07, ÅGJ08]: this paper departs from the same idea as ours: strategies assigned to agents by strategy quantifiers are not dropped when evaluating the subformulas. There are however two main differences with our approach: first, the authors mostly restrict to memoryless strategies, which makes the problem much easier as the number of such strategies is finite. Second, strategies cannot be dropped from the context: actually, subformulas under a strategy quantifier are evaluated in the subtree corresponding to the outcomes of the selected strategy. Model-checking ATL with memoryless irrevocable strategies is PSPACE-complete [ÅGJ08].
- Stochastic Game Logic [BBGK07]: this approach consists in evaluating subformulas in the subtree containing all the outcomes of already selected strategies. This way of evaluating subformulas within a *strategy context* is precisely the approach we have adopted and we describe below. However, in [BBGK07], this semantics is studied in a probabilistic setting, namely over turn-based games that contain stochastic states, equipped with a probability distribution over their successors. It is proved that model checking is undecidable in the general framework, but becomes decidable when restricting to (mixed or deterministic) memoryless strategies.
- Strategy Logic [CHP07]: the approach in Strategy Logic is different: it extends LTL with first-order quantification over strategies. An ATL formula of the form $\langle\langle A \rangle\rangle \varphi$ would then be written as $\exists x_A. \forall x_B. \varphi(x_A, x_B)$. Several strategies can then be used at different places in the formula, making this formalism very expressive. However, it has several restrictions: it is defined only in the setting of two-player turn-based games, and more importantly, it does not allow the nesting of non-closed

subformulas in the scope of temporal modalities. Hence a formula such as (4.2) (having the *strategy-context* semantics in mind) would not be expressible (at least not directly) in Strategy Logic³. Still, SL encompasses ATL*, and can also express non-zero-sum objectives such as Nash equilibria or dominance of strategies.

Strategy-Logic model checking is proved decidable (with a tree-automata based approach, whose complexity is non-elementary).

- **QD μ** [Pin07]: QD μ extends the μ -calculus with two important features. First, *decision modalities* allow to select some of the successors of the current state, namely those that correspond to a move of a player: formula $\Diamond_A(q)$ states that there is a move of Player A whose (one-step) outcomes are exactly the successors labelled with atomic proposition q . This is further extended with *quantification* over atomic propositions on the tree. QD μ can express SL and much more (thanks to fixpoint constructions), and is defined over n -player CGSs. A non-elementary algorithm is proposed in [Pin07], based on alternating tree automata. This algorithm is correct only for a subclass of CGSs⁴.

4.2.1 Expressiveness of ATL with strategy contexts

Our approach corresponds to the second one, but restricted to non-stochastic games (and pure strategies): syntactically, our new logic (which we called ATL_{sc} [BDLM09]) is similar to ATL, but we replaced the strategy quantifier $\langle\langle A \rangle\rangle$ with $\langle A \rangle$, to mark the difference. Semantically, ATL_{sc} formulas are evaluated within a context. This modifies the semantics as follows:

- a strategy quantifier $\langle A \rangle$ now stores the witnessing strategy for A into the context. If some agents in A were already assigned a strategy in the context, this older strategy is replaced with the new one:

$$\mathcal{G}, \rho \models_F \langle A \rangle \varphi_p \quad \text{iff} \\ \exists F_A \in \text{Strat}(A). \quad \forall \rho' \in \text{Out}^\infty(\text{first}(\rho), F_A \circ F). \quad \mathcal{C}, \rho' \models_{F_A \circ F} \varphi_p$$

where $F_A \circ F$ is the new strategy context, mapping players in A to their strategy in F_A and players in $\text{dom}(F) \setminus A$ to their strategy in F .

- when evaluating a temporal modality, apart from the corresponding condition to be fulfilled, we have to “update” the strategy context according to the run of the automaton.

$$\mathcal{G}, \rho \models_F \varphi_p \mathbf{U} \psi_p \quad \text{iff} \\ \exists i \geq 0. \quad \mathcal{C}, \rho^{i \rightarrow \infty} \models_{F \rho^{0 \rightarrow i}} \psi_p \quad \text{and} \quad \forall 0 \leq j < i. \quad \mathcal{C}, \rho^{j \rightarrow \infty} \models_{F \rho^{0 \rightarrow j}} \varphi_p$$

³In [MMV10], Strategy Logic is extended to a more general setting where both restrictions are dropped. See Section 4.2.3 for a discussion on this extension.

⁴Personal communication with Sophie Pinchinat. The subclass corresponds to *alternating transition systems* [AHK98], which enjoy the property that $\text{Next}(s, C \cup C', m_C \oplus m_{C'}) = \text{Next}(s, C, m_C) \cap \text{Next}(s, C', m_{C'})$ for any disjoint coalitions C and C' and move vectors m_C and $m_{C'}$.

where at each step, the context is updated as $F^{\rho^{0 \rightarrow i}}$, which is defined by $F^{\rho^{0 \rightarrow i}}(A)(\lambda) = F(A)(\rho^{0 \rightarrow i} \cdot \lambda)$.

Regarding expressiveness, this extension comes with a much enhanced expressive power, both on a practical and theoretical sense. On the practical side, ATL_{sc} provides a way to express non-zero-sum winning conditions. As a classical example of this, we mention (Boolean, pure-strategy) Nash equilibria: ATL_{sc} can express the existence of such Nash equilibria as follows:

$$\langle A_1, \dots, A_k \rangle \bigwedge_{1 \leq i \leq k} \left(\langle A_i \rangle \varphi_i \Rightarrow \varphi_i \right). \quad (4.3)$$

This formula states that there is a strategy profile in which no player can improve her payoff. ATL_{sc} can also express interactions between a server and different clients trying to access to a shared resource. The objective of the server is two-fold: ensure mutual exclusion first, so that only one client can access the resource at a time; and also ensure that each client can access the server if they ask for it. This would be stated as follows:

$$\langle \text{Server} \rangle \mathbf{G} \left[\neg \bigwedge_{C \neq C' \in \text{Clients}} (\text{access}_C \wedge \text{access}_{C'}) \quad \wedge \quad \bigwedge_{C \in \text{Clients}} \langle C \rangle \mathbf{F} \text{access}_C \right] \quad (4.4)$$

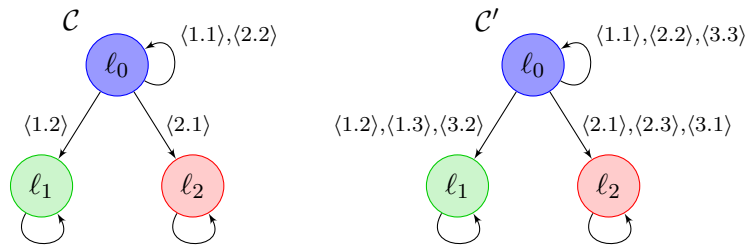
On the theoretical side, we proved the following results:

Theorem 31 ([BDLM09, DLM10]). *ATL_{sc} and ATL_{sc}^* are equally expressive, and they are strictly more expressive than ATL^* and GL .*

In this statement, GL is the *game logic* [AHK02], especially useful for encoding the *module-checking* problem [KVW01]. In GL , strategy quantification is disconnected from (*e.g.* CTL-like) properties of the resulting execution tree. This clearly encompasses ATL^* , in which strategy quantification is coupled with universal path quantification.

The fact that ATL_{sc} is highly expressive is already witnessed by the fact that it can distinguish between games that are *alternating-bisimilar* [AHKV98]. Alternating bisimulation (see Section 1.1.2) extends classical bisimulation for games models: an alternating bisimulation is a relation between states such that for two states that are in the relation, a move of any coalition from one of the states can be mimicked by a move of the same coalition from the other state, reaching only related states. In other terms, with any “winning” move from one of the two related states, we can associate a “winning” move from the other state.

For instance, the initial states of \mathcal{C} and \mathcal{C}' on the right are alternating-bisimilar, as witnessed by the obvious relation between states of \mathcal{C} and states of \mathcal{C}' : it is easy to see that, for both players, moves 1 and 2 are



equivalent from ℓ_0 and ℓ'_0 . The case of move 3 is a bit different: since for each player, playing move 3 may lead to all three states, it can be mimicked by any move from ℓ_0 in \mathcal{C} . Now, it is easily seen that formula $\langle A_1 \rangle (\langle A_2 \rangle \mathbf{X} \ell_1 \wedge \langle A_2 \rangle \mathbf{X} \ell_2)$ only holds in the initial state of \mathcal{C}' , which proves the statement above.

4.2.2 Model checking algorithm for **ATL*** with strategy contexts

Several tree-automata-based model-checking algorithms have been proposed for related logics:

- **ATL*** model checking is achieved by labelling states of the considered CGS with the subformulas they satisfy. In the end, the algorithm consists in checking states from which a given coalition has a strategy for fulfilling an LTL objective φ . This is achieved by building a Büchi tree automaton for the LTL formula, and turning it into a deterministic Rabin tree automaton accepting trees all of whose branches satisfy φ . On the CGS side, the algorithm consists in building a Büchi tree automaton accepting those trees that represent (exactly) the outcomes of a strategy of coalition A from ℓ_0 . The product of both automata is non-empty iff state ℓ_0 satisfies $\langle\langle A \rangle\rangle \varphi$.
- the **QD μ** model-checking algorithm consists in associating an alternating parity tree automaton with a **QD μ** formula. This is achieved bottom-up on the formula, using projection to encode existential quantification over propositions. Since projection requires turning the alternating automaton into a non-deterministic one, the algorithm has non-elementary complexity ($k+1$ -EXPTIME where k is the maximal number of nested quantifiers).
- **SL** model checking follows similar ideas. Here, subformulas in the scope of temporal modalities are state formulas, and can be handled inductively in a labelling algorithm. It then suffices to build the automata for the LTL subformulas, and apply conjunction or disjunction of automata to handle conjunction or disjunction of subformulas, and projection to handle quantification over strategies. As previously, this requires turning the alternating automaton into a non-deterministic one, except for consecutive quantifications of the same type, where the nature of the automaton can be preserved. In the end, the algorithm runs in $k+1$ -EXPTIME where k is the maximal number of quantifier alternations.

Our approach resembles the last two approaches, in that it builds an alternating parity tree automaton by induction on the structure of the formula. One main difference, however, is that we handle the full class of n -player CGSs, while the algorithm for **SL** in [CHP07] only handles two-player turn-based games and the one for **QD μ** only works for a subclass of CGSs ⁵.

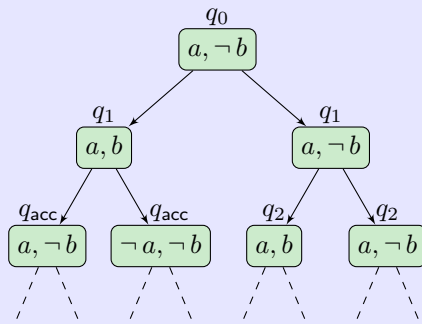
In order to handle the full class of CGSs, our algorithm works on the unwinding of the CGS from a given state q , where the formula is to be checked. This unwinding is an

⁵Recently, an algorithm for **SL** on the full class of n -player CGSs has been proposed [MMV10]. We come back on this algorithm in the next section.

Tree automata extend classical (word) automata with the ability to read tree as input and send different computations along the different branches of the input tree.

Formally, a Λ -tree automaton over Σ is a finite-state automaton $\mathcal{A} = \langle Q, q_0, \delta, F \rangle$ where Q is the finite set of states of the automaton, $q_0 \in Q$ is the initial state, $\delta: Q \times \Sigma \rightarrow 2^{\Lambda \rightarrow Q}$ is the transition function, and $F: Q^\omega \rightarrow \{0, 1\}$ specifies the acceptance condition.

Given an input Σ -labelled Λ -tree \mathcal{T} , a run of \mathcal{A} on \mathcal{T} is a Q -labelled Λ -tree \mathcal{U} , with the same structure as \mathcal{T} but with a different labelling: the root is labelled with q_0 , for any node n labelled with σ in \mathcal{T} and with q in \mathcal{U} , the successor nodes $n \cdot \lambda_i$ of n in \mathcal{U} are labelled according to one of the functions $t: \Lambda \rightarrow Q$ returned by $\delta(q, \sigma)$. The input tree \mathcal{T} is accepted by \mathcal{A} if there exists a run of \mathcal{A} over \mathcal{T} all of whose branches satisfy the acceptance condition.



Example. Consider the following four-state Λ -tree automaton over $2^{\{a,b\}}$ defined as $\mathcal{A} = \langle \{q_0, q_1, q_2, q_{acc}\}, q_0, \delta, F \rangle$ where the transition function is defined as

$$\begin{aligned} \delta(q_0, (\neg a, \neg b)) &: \lambda \mapsto q_{acc} & \delta(q_0, (\neg a, b)) &: \lambda \mapsto q_{acc} \\ \delta(q_0, (a, \neg b)) &: \lambda \mapsto q_1 & \delta(q_0, (a, b)) &: \lambda \mapsto q_1 \\ \delta(q_1, (\neg a, \neg b)) &: \lambda \mapsto q_{acc} & \delta(q_1, (\neg a, b)) &: \lambda \mapsto q_{acc} \\ \delta(q_1, (a, \neg b)) &: \lambda \mapsto q_2 & \delta(q_1, (a, b)) &: \lambda \mapsto q_{acc} \\ \delta(q_2, (\neg a, \neg b)) &: \lambda \mapsto q_{acc} & \delta(q_2, (\neg a, b)) &: \lambda \mapsto q_{acc} \\ \delta(q_2, (a, \neg b)) &: \lambda \mapsto q_2 & \delta(q_2, (a, b)) &: \lambda \mapsto q_2 \end{aligned}$$

The acceptance function returns 1 on all branches that eventually reach q_{acc} . It can be checked that this automaton implements the CTL* formula $\mathbf{A}(\mathbf{G} a \Rightarrow \mathbf{X} b)$. A similar automaton could be built for checking formula $\mathbf{A}(\mathbf{G} a \Rightarrow b \mathbf{U} c)$.

Alternating tree automata extend tree automata with the ability to fork several simultaneous executions along the same direction: the transition function then maps $Q \times \Sigma$ to $2^{\Lambda \rightarrow (2^Q \setminus \emptyset)}$. The execution tree may then contain several copies of the same branch, corresponding to the simultaneous executions that are forked along that branch. Acceptance of an execution tree is still defined as acceptance of all the branches.

Under reasonable acceptance conditions (for instance, the parity acceptance condition [EJ91]), alternating tree automata enjoy the following properties:

- they are closed under union, intersection and complement, with effective (and linear-size) constructions [MS87, MS95];
- alternating tree automata can be translated into equivalent non-deterministic tree automata, with an exponential blow-up [MS95];
- given a non-deterministic tree automaton \mathcal{A} and a given atomic proposition p such that $\Sigma = \Sigma' \times \{p, \neg p\}$, one can build a linear-size non-deterministic tree automaton \mathcal{B} which accepts exactly the Σ' -labelled trees that can be labelled with p in order to be accepted by \mathcal{A} [MS85]. In other terms, \mathcal{B} accepts the projection on Σ' of the trees accepted by \mathcal{A} .
- alternating tree automata emptiness can be decided in exponential time [EJ91, EJ99].

Table 4: Alternating Parity Tree automata

infinite tree whose directions are the set of states of the CGS⁶. Each node represents a finite play of the CGS, and is labelled with the following information:

- information about the corresponding state of the CGS, *i.e.*, its name, labelling, and transition table;
- for each player, one selected move or the special symbol \perp . This will be used to store selected strategies of the players, if any.
- extra atomic propositions p_{Out} , p_{Left} and p_{Right} used for technical purposes.

The first part of this labelling are given by the CGS, and is fixed all along the computation. The second item is the way we encode strategies in the tree; \perp indicates that the corresponding player is not assigned a strategy in the corresponding node. We will assume that each player is either assigned a strategy in all nodes, or in none; this can be checked by an intermediary tree automaton. The last three atomic propositions are used to label nodes that belong to outcomes of the selected strategies (p_{Out}), and nodes at which the left-hand side- or right-hand-side subformulas of a modality hold. We write $\Sigma_{\mathcal{C}}$ for the alphabet corresponding to the first item of this labelling (which actually contains at most as many letters as the number of states in the CGS), and $\Sigma_{\mathcal{C}}^+$ for the whole alphabet (which has size exponential due to the labelling with strategies). Notice that given a CGS \mathcal{C} and a state ℓ_0 , it is easy to come up with a deterministic tree automaton accepting only the unwindings of \mathcal{C} from ℓ_0 (they may differ on their labellings for $\Sigma_{\mathcal{C}}^+ \setminus \Sigma_{\mathcal{C}}$).

Given such an unwinding \mathcal{T} labelled as above, we write $\text{Strat}_D^{\mathcal{T}}$ for the strategies of players in D it encodes. Again, we may assume that all players in D are assigned a strategy in the tree, by adding an intermediary tree automaton for checking this.

With an ATL_{sc} formula φ , we associate an alternating parity tree automaton that accepts the trees encoding the unwinding of a CGS from a state satisfying φ . The construction is inductive. Here we only briefly explain how to build an automaton $\mathcal{A}_{\langle A \rangle \mathbf{X} \varphi, D}$ for $\langle A \rangle \mathbf{X} \varphi$ under a context involving a set D of players from an automaton $\mathcal{A}_{\varphi, D \cup A}$ for φ under a context involving players in $D \cup A$. This case already contains the main ideas of our proof.

- As a first step, we build an automaton accepting trees in which the subtrees rooted at nodes labelled with p_{Right} are accepted by $\mathcal{A}_{\varphi, D \cup A}$. Hence nodes labelled with p_{Right} are nodes where φ holds under the current context for $D \cup A$.
- Using classical tree-automata approaches for CTL^* , we build an automaton accepting those trees in which formula $\mathbf{A}(\mathbf{G} p_{\text{Out}} \Rightarrow \mathbf{X} p_{\text{Right}})$. Since p_{Out} will label the outcomes of selected strategies, this automaton combined with the previous one will precisely ensure that all the outcomes satisfy $\mathbf{X} \varphi$, where φ is evaluated under the context for coalition D .
- The third step is to build an automaton accepting trees in which all the nodes appearing in outcomes of the strategies of players in $A \cup D$ are labelled with p_{Out} .

⁶It would be more natural to take for directions the set of move vectors, but strategies would then depend on the moves played along the run instead of the sequence of visited states.

- Finally, we project away the strategy for A , hence getting an automaton accepting those trees that *can* be labelled with strategies for players in A in such a way that all the outcomes satisfy $\mathbf{X}\varphi$.

Notice that the last step requires turning the alternating tree automaton into a non-deterministic one, involving an exponential blowup. In the end, we get:

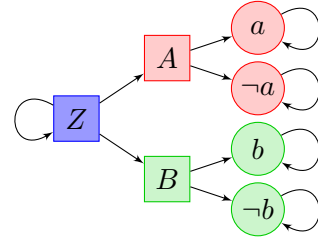
Proposition 32. *Given an ATL_{sc} formula φ , a CGS \mathcal{C} and a state ℓ_0 of \mathcal{C} , we can build an alternating parity tree automaton \mathcal{A} s.t. $\text{Lang}(\mathcal{A})$ is non-empty if, and only if, $\mathcal{C}, \ell_0 \models_{\emptyset} \varphi$. Moreover, \mathcal{A} has size k -exponential and index $(k - 1)$ -exponential, where k is the number of nested strategy quantifiers.*

Since emptiness for alternating parity tree automata can be checked in exponential time, we end up with a $(k + 1)$ -exponential time algorithm for model checking ATL_{sc} . Handling ATL_{sc}^* is a straightforward extension of the above algorithm, which does not modify the overall complexity:

Theorem 33 ([DLM10]). *ATL_{sc} and ATL_{sc}^* model checking can be achieved in $(k+1)$ -EXPTIME, where k is the maximal number of nested strategy quantifiers.*

Recently, we managed to prove that our algorithm is essentially optimal, by showing that the problem cannot have an elementary solution. This is obtained quite easily by encoding the satisfiability problem for the logic QPTL [SVW87], an extension of LTL with quantification over the labelling of atomic propositions on infinite runs. For example, formula $\forall a. \exists b. \mathbf{G}(b \Leftrightarrow \mathbf{X}a)$ states that for any labelling of an infinite run with atomic proposition a , it is possible to label the same run with atomic proposition b in such a way that each b is followed by an a . Our encoding is as follows:

given a formula with n quantifier atomic propositions, we consider the $n + 1$ -player turn-based game depicted on the above, where capital letters are the names of the players, and lower-case letters are atomic propositions. In this game, player Z can loop in the initial location forever (this special run corresponds to the infinite run to be labelled in QPTL), and at each round, he can “ask” the other players for choosing the value of the atomic proposition they control. This way, the above QPTL formula would be expressed as follows in ATL_{sc}^* :



$$[[A] \langle B \rangle \left[\mathbf{G} \langle Z \rangle \Rightarrow \mathbf{G} (\langle Z \rangle \mathbf{X} \mathbf{X} \langle b \rangle \Leftrightarrow \mathbf{X} \langle Z \rangle \mathbf{X} \mathbf{X} \langle a \rangle) \right].$$

Theorem 34 (Unpublished). *ATL_{sc} model-checking is k-EXPSpace-hard for formulas with $k + 1$ nested strategy quantifiers.*

Remark. Notice that, contrary to SL, ATL_{sc} may have implicit quantifiers (which we don't have to consider when computing the number of nested quantifiers). For instance, formula $\langle\langle A \rangle\rangle \varphi$ has only one quantifier, while the equivalent SL formula has two (and one alternation): $\exists x_A. \forall x_B. \varphi(x_A, x_B)$.

4.2.3 A note on strategy logic

Very recently, an elementary (2-EXPTIME) algorithm has been proposed for an extension of SL [MMV10]. This extension has no restriction on nested subformulas, and is defined on the whole class of n -player CGSs. As a consequence, it encompasses ATL_{sc} (with a linear translation), which yields a contradiction with our k-EXPSpace-hardness result.

It seems that the algorithm in [MMV10] only handles *restricted* Strategy Logic, where subformulas nested under temporal modalities must be closed (this restriction is present in the definition of SL from [CHP07]): this way, subformulas are really *state-formulas*, and the labelling algorithm of [MMV10] can be applied. The main idea of the algorithm is then to turn all strategy quantifications in the transition function of the alternating tree automaton. This algorithm thus provides an important improvement on the algorithm proposed in [CHP07] for Strategy Logic with restricted nesting, but is not valid for the unrestricted version of the logic.

Our algorithm can easily be extended to handle *unrestricted* Strategy Logic, since our tree automata could store more than one strategy per agent. In the end:

Theorem 35 (Unpublished). *Strategy-Logic model checking is decidable, with non-elementary complexity.*

Notice that our reduction to QPTL above could be adapted to the setting of two-player turn-based games (by letting one player control all the atomic propositions, with one strategy per proposition), meaning that the blow-up in complexity only comes from nesting non-closed subformulas.

Finally, the satisfiability problem for Strategy Logic is addressed in [MMV10], where it is proved undecidable over n -player CGSs. Whether or not it is decidable on turn-based games remains open. Similarly, whether or not ATL_{sc} satisfiability is decidable is part of our future work.

4.3 Really concurrent timed game automata

In the timed setting, games are inherently concurrent: the date of the next action can usually be chosen concurrently by the players. The classical model of *timed game*

automata [AMPS98, dAFH⁺03] runs as follows⁷: the set of transitions is partitioned among the (two) players, and at each step, each player chooses a delay and one of her allowed transitions. This represent an infinite-state CGS where the (infinite) set of moves (which we call *timed moves*) is the set of pairs (τ, δ) where τ is the proposed delay to be elapsed before firing transition δ . It is required that if a player chooses a move (τ, δ) , then τ must be small enough to not violate the invariant in the current state, δ must belong to that player, and the guard of δ as well as the invariant in the target state must be fulfilled. Once both players have selected a move, the player with the shortest delay is elected and can then apply her choice (both the delay and the transition). On the other player, which did not select the shortest delay, has no influence on the transition.

The existence of winning strategies for such games is decidable, provided that the objectives are reasonable. More precisely, it can be proved that the set of controllable predecessors of a union of regions is also a union of regions [MPS95]. It follows that the analysis of timed games can be done on the (finite-state) region game, provided that the objectives are region-definable. Parity winning conditions can then be checked in deterministic exponential time. This setting has also been extended to compute *non-Zeno* strategies [dAFH⁺03]. We refer to Section 4.3.4 below, where we adapted this technique in our setting.

4.3.1 Timed concurrent game structures

We have proposed a *more concurrent* model of timed games, where players not only propose a delay and a transition to be taken after that delay, but also a function indicating the action they want to play if an earlier move is proposed by some other player. In other terms, all the players have a say in the choice of the transition to be taken, even if they did not propose the shortest delay. In this setting, a *timed move* is a pair (τ, σ) where τ is the preferred delay at which the player want the next transition to take place, and σ maps all intermediate delays to the action this player would play if the transition were to be taken earlier.

Defining this model is a bit technical: since the transition to be taken is chosen concurrently, it cannot be assumed that its guard is satisfied at the time it has to be fired. The timed model underlying our concurrent timed game automata is thus a bit different from timed automata, in that a “transition” now should map any clock valuations to a state and a set of clocks to be reset: in other terms, any transition should be available at any time, but possibly with different target states and resetted clocks. As for symbolic CGSs, we define this new kind of transitions symbolically:

Definition 36. *A generalized timed automaton is a 4-tuple $\mathcal{A} = \langle \mathcal{S}, \mathcal{X}, \text{cond}, \text{inv} \rangle$ similar to classical timed automata (Section. 1.3.1), with the exception that cond maps each location to a finite set of finite sequences $(\varphi_i, r_i, \ell_i)_{i \in I}$ where for all i , $\varphi_i \in T\text{Constr}(\mathcal{X})$, $r_i \subseteq 2^{\mathcal{X}}$ indicates the clocks to be reset, and ℓ_i is the target location.*

⁷Another semantics has been proposed in the literature, in which the semantics does not explicitly deals with the strategy of the antagonist: a strategy σ for Player A is winning if all the runs that are compatible with this strategy meet the objective. This approach was defined in [MPS95] and is used *e.g.* in [CDF⁺05].

In this setting, the transitions in the transition system \mathcal{S} are named *edges*, and each sequence in each $\text{cond}(s)$ is called a *transition*. A transition may say for instance that “if clock x is in $[2, 4]$ and y is less than 1 then go to s' else stay in s ”.

Our timed concurrent game automata extend this model with players. Once again, the transition table associating a transition to a move vector is encoded symbolically, using constraints $\text{MConstr}(\text{Agt}, \text{Mov})$ as defined on page 76. The semantics of these constraints is similar to the one for symbolic CGSs, with the exception that constraint $A = m$ here means that player A has played move m at date τ_{\min} , which is the value of the smallest delay among the delays chosen by the players.

We are now ready to put everything together in our definition:

Definition 37. A timed CGS is a 8-tuple $\mathcal{G} = \langle \mathcal{S}, \mathcal{X}, \text{cond}, \text{inv}, \text{Agt}, \text{Mov}, \text{Chc}, \text{Edg} \rangle$ where $\langle \mathcal{S}, \mathcal{X}, \text{cond}, \text{inv} \rangle$ is a generalized timed automaton, Agt is a finite set of players with moves in Mov whose availability is given by $\text{Chc}: S \times \text{Agt} \rightarrow 2^{\text{Mov}} \setminus \emptyset$, and where for each location ℓ , $\text{Edg}(\ell)$ is a finite sequence $(\varphi_i, t_i)_{i \in I}$ where for all $i \in I$, φ_i is a formula in $\text{MConstr}(\text{Agt}, \text{Mov})$ and t_i is a transition of the generalized timed automaton.

In this setting, a move for Player A is a pair made of a delay τ , corresponding to the delay at which this player would prefer the next action to be played, and of a mapping σ from $[0, \tau]$ to allowed moves in the current state. Notice that our model does not encompass the classical model of timed game automata [AMPS98, dAFH⁺03]: indeed, in our model, we have no way of keeping track of the player who proposed the shortest delay. We address this problem in Section 4.3.5, by proposing a slight extension of timed CGSs.

4.3.2 Properties of strategies in timed CGSs

While we have increased the expressive power of the model, we did not introduce too much power in terms of the timing constraints, so that timed CGSs still enjoy the nice property of being *region-compliant*: we prove that region-based strategies are sufficient for region-based objectives. Being region-based is two-fold:

- a strategy is said *region-invariant* when its values on region-equivalent histories are equivalent. This notion is formally defined in terms of *isomorphisms* between region-equivalent configurations. Roughly, such an isomorphism *translates* the delays from one configuration to the other in order to make the delayed configurations region-equivalent. Fig. 4.2 depicts an isomorphism between two equivalent configurations (q, v) and (q', v') . This is a special case of a piecewise-affine isomorphism, which we call *canonical*.

A strategy is region-invariant when its values after two region-equivalent histories are related by an isomorphism. Two equivalent histories then give rise to equivalent outcomes when this property is met.

- a strategy is said *region-uniform* when for any finite history ρ , its second component σ is constant on each region visited when letting time elapse from $\text{last}(\rho)$.

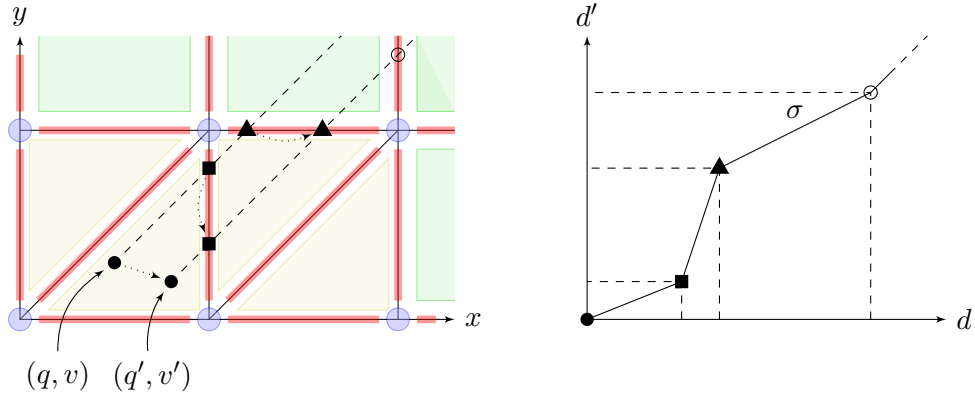


Figure 4.2: Isomorphism between two equivalent configurations

The important technical result is that winning strategies for region-defined objectives can be made region-invariant and region-uniform. The resulting strategies are then fully region-based in the sense that they only depend on the set of regions visited in the history and return region-definable values. This allows us to work on the finite abstraction obtained by quotienting the timed CGS by the region equivalence. In this finite CGS, moves of the players are pairs made of an integer (corresponding to the proposed delay, but expressed in terms of regions being visited) and a function mapping all intermediate regions to an action. Not surprisingly, we get:

Theorem 38 ([BLMO07]). *Region equivalence induces an alternating bisimulation between the infinite-state CGS associated with a timed CGS and the finite-state region CGS.*

There is of course a blowup in the size of this *region CGS*, compared to the size of the original timed CGS: the number of states is exponential, as is the case for the region automaton associated with a timed automaton. The number of moves is doubly-exponential (in the size of the maximal constant), since a move in the region CGS contains a function mapping the time-successor regions of the current region to a move. The size of the transition table also includes this doubly-exponential blowup. Even if we manage to preserve the symbolic encoding of the transition table during the translation, we still have a doubly-exponential blowup due to the fact that we have to “find” the earliest regions among the choices of all players.

4.3.3 Timed ATL model checking on timed CGSs

ATL and ATL* can be extended with timing constraints, in pretty much the same way as TCTL extends CTL. As was the case in Chapter 2, the extension can either decorate modalities with timing constraints (as we defined and studied in [LMO06] over weighted CGSs) or involve formula clocks (as defined and studied in [HP06] over timed game automata). In this section, we consider the latter approach, with formula clocks

to measure time lapses between different events. The syntax is as follows:

$$\begin{aligned} \text{TATL}^* \ni \varphi_s &::= p \mid c \sim n \mid \neg \varphi_s \mid \varphi_s \vee \varphi_s \mid \langle\langle A \rangle\rangle \varphi_p \\ \varphi_p &::= \varphi_s \mid c.\varphi_p \mid \varphi_p \wedge \varphi_p \mid \varphi_p \vee \varphi_p \mid \varphi_p \mathbf{U} \varphi_p \mid \varphi_p \mathbf{R} \varphi_p \end{aligned}$$

where c ranges over a finite set of formula clocks. The semantics is as expected, assuming strict semantics for “until” and “release” modalities, as is usual in the timed case.

Since this logic encompasses TPTL, there is no hope of getting a model-checking algorithm for it. On the other hand, the classical restriction TATL to unnested temporal modalities is clearly decidable: it suffices to apply a classical ATL labelling algorithm on the region CGS (extending the region equivalence relation with formula clocks). Because of the doubly-exponential blowup in the region CGS, a direct implementation of the labelling algorithm is in 2-EXPTIME. It is possible however to implement the algorithm in a space-efficient manner, which makes it run in EXPSPACE⁸.

We have proposed an intermediary logic, which preserves decidability and can express Büchi conditions, which we will use in the next section to rule out *Zeno* strategies. This logic combines TATL with (untimed) LTL. Formally:

$$\begin{aligned} \text{TALTL} \ni \varphi_s &::= p \mid c \sim n \mid \neg \varphi_s \mid \varphi_s \vee \varphi_s \mid \langle\langle A \rangle\rangle \varphi_p \mid c.\varphi_s \\ \varphi_p &::= \varphi_s \mid \varphi_p \wedge \varphi_p \mid \varphi_p \vee \varphi_p \mid \varphi_p \mathbf{U} \varphi_p \mid \varphi_p \mathbf{R} \varphi_p \end{aligned}$$

The only difference with TATL* is in the “freeze quantifier” $c.\varphi_s$ being followed by a state formula. This way, freeze quantifiers can only appear together with a strategy quantifier. On the other hand, timing constraints are always allowed at any point in the path formulas. Decidability is then obtained by adding formula clocks in the region equivalence, and applying the classical labelling algorithm for ATL* formulas on the region CGS. Hence:

Theorem 39 ([BLMO07]). *Model checking TATL over timed CGSs is in EXPSPACE⁸, and EXPTIME-hard. Model checking TALTL over timed CGSs in 2-EXPTIME-complete.*

4.3.4 Ruling out Zeno strategies

Zeno strategies are unrealistic strategies that consist in proposing a time-converging sequence of delays, thus enforcing safety by preventing that a guard of the form $x \geq c$ is ever met. This is a special case of *non-robust* behaviour, has the ones depicted in Section 2.3. While the synthesis of robust strategies is still a very challenging problem, ruling out Zeno strategies can be achieved in our timed CGSs, following ideas from [dAFH⁺03]. The idea is the following: each time a player has selected the shortest delay, she is assigned a blame (hence, contrary to timed game automata, several player can be given a blame at the same time). For a strategy to be winning, it has to reach the winning objective of the player, but it must also have the following property: along any time-convergent outcome (if any), the player must receive finitely many blames.

⁸This corrects our claim in [BLMO07] that TATL model-checking is in EXPTIME over timed CGSs.

Keeping track of blames and time divergence can be achieved on top of the region CGS: time divergence is checked with an extra clock that is reset each time it reaches value 1. An extra atomic proposition `tick` is turned on each time this clock is reset, and time divergence is then equivalent to infinitely many occurrences of this `tick` event along the outcome. Blames are handled by taking 2^{Agt} copies of the region CGS, each copy being labelled with atomic propositions `blamei` for each player to be blamed after the previous move. We may then define non-Zeno strategy quantifiers $\langle\langle A \rangle\rangle_{\text{non-Zeno}}\varphi$, which can be checked thanks to the following equivalence:

$$\langle\langle A \rangle\rangle_{\text{non-Zeno}}\varphi \equiv \langle\langle A \rangle\rangle \left(\varphi \vee \left(\neg \mathbf{F}^{\infty} \text{tick} \wedge \bigwedge_{a \in A} \neg \mathbf{F}^{\infty} \text{blame}_a \right) \right)$$

which is a TALTL formula (provided that φ is). Non-Zeno strategy quantifiers in TALTL formulas can be handled more efficiently by relying on the algorithm for FairATL. In the end:

Theorem 40 ([BLMO07]). *Model checking $TATL_{\text{non-Zeno}}$ over timed CGSs is in EXPSPACE, and EXPTIME-hard. Model checking $TALTL_{\text{non-Zeno}}$ over timed CGSs in 2-EXPTIME-complete.*

4.3.5 Extensions of the model

timed CGSs are not powerful enough to express the classical model of timed games: indeed, the constraints in MConstr cannot behave differently for the player(s) who selected the shortest delay. Our aim, which is sketched in [Ore08] and in preparation as a long version of [BLMO07], is to extend the constraints in timed CGSs. A simple way of checking whether a player has selected the shortest delay is to allow tests of the form $\tau_A = \tau_{\min}$ (with the intended meaning that Player A has selected the shortest delay). Using such constraints (and allowing non-determinism when several players select the minimal delay, which would not modify our algorithms very much), timed CGSs would be able to encode timed game automata, with also the possibility to give priority to some players depending on the state.

4.4 Conclusions and future works

My main result in the study of concurrent games and related temporal logics in the untimed case is the definition of ATL_{sc} , an extension of the semantics of ATL that is well-suited for non-zero-sum games. Unfortunately, the model-checking problem for this logic has non-elementary complexity (but interesting/understandable formulas will often have only few nested quantifiers).

These results are very recent, and not everything is well understood in this setting. ATL_{sc} and SL seem to be closely related to the logic QCTL [Kup99, Fre06], extending CTL with quantification on atomic propositions (over execution trees). Such quantification is precisely what we need to encode strategy quantification. It would be nice to check how

tight the link is, and whether it can give new results (especially for satisfiability of the logics).

Other interesting problems in the area include the design of a behavioural equivalence corresponding to ATL_{sc} : this logic can distinguish between two alternating-bisimilar CGSs, and a stronger equivalence relation is needed. On another note, contrary to ATL , ATL_{sc} strategies require memory. Evaluating how much memory is needed to achieve some goal would be interesting. Notice that we already studied the model-checking problem for ATL_{sc} with bounded-memory quantifiers, which easily reduces to guessing memoryless strategies, and can then be solved in exponential time. It follows that in the general case, the complexity of strategies for ATL_{sc} may be non-elementary. Finally, introducing probabilistic strategies in the setting of non-zero-sum concurrent games would be a very exciting research direction. Unfortunately, the results of [BBGK07] leaves little hope for a general and decidable solution, even if the game is deterministic. Restricted fragments (*e.g.* ATL_{sc} or SL with restricted nesting) may offer decidable alternatives.

In the second part of this chapter, I presented an extension of CGSs with real-time. This extension is rather different from classical timed games, which are more of an extension of timed automata with two players. We proved that this new model enjoys similar properties as timed game automata of [dAFH⁺03], especially that they can be handled using the region abstraction.

Recently, with Patricia Bouyer-Decitre and Romain Brenguier, we started studying non-zero-sum games in the timed setting (using standard timed game automata). Our current results concern Nash equilibria for reachability objectives (in pure strategies). We developed the notion of *repellor set*, which, for a coalition A , is a set of states from which there is a strategy profile in which players in A will lose, even if any of them (unilaterally) changes her strategy. If the outcome of the strategy profile also visits the objectives of players not in A , then this is a Nash equilibrium. This characterization can be applied to finite or infinite concurrent games. Algorithmically, we proved that it provides NP algorithms for finite-state concurrent games, and EXPTIME algorithms for timed games, via a reduction to the (slightly modified) region game.

Theorem 41 ([BBM10b]). *The existence of pure-strategy Nash equilibria with reachability objectives over timed game automata is EXPTIME-complete.*

In the case of two players, using different techniques (transforming the timed game under study into two finite-state turn-based games), we were able to extend this result to ω -regular objectives [BBM10a].

All these results open many research directions. One of them concerns the extension to timed CGSs of the results existing for timed game automata. In particular, I believe that timed CGSs are well-suited for several problems in the verification of real-time open systems, *e.g.* for robust synthesis. timed CGSs would also simplify (conceptually) the study of Nash equilibria in timed games, as the non-determinism inherent in timed game automata requires adapting the notion of Nash equilibria to non-deterministic games. Another interesting of the above results is the extension to probabilistic strategies, which

are especially interesting in concurrent games with non-zero-sum objectives. Extending timed games with probabilistic strategies (on the delays as well as on the transitions) appears to be a very exciting, but also very challenging, research direction.

Conclusions and perspectives

This thesis summarizes my main contributions in the study of formal verification for open and timed systems. In the setting of timed systems (Chapter 2), I studied the expressiveness and model-checking problem of various timed temporal logics over timed automata, defining in particular a new timed temporal logic extending MITL with punctual constraints while remaining decidable. For timed automata extended with hybrid observers (Chapter 3), I proved several undecidability results in the general case, and decidability results when restricting to one-clock weighted timed automata. I also introduced energy constraints in that setting, and again obtained various decidability and undecidability results. Finally, I extended ATL in the setting of concurrent games (Chapter 4), obtaining a very rich temporal logic for expressing non-zero-sum properties while remaining decidable.

At the end of each chapter, I already outlined some possible research directions. I briefly summarize them below.

Timed automata with observers. The extension of timed automata with observers provides a very convenient model for representing various resource-management problems of embedded systems. Indeed, resource consumption usually crucially depends on time (making timed automata a very convenient underlying model for this problem), and timed automata with hybrid observers offer some decidability problems, especially optimal reachability (and optimal reachability under *lower-bound energy constraint* under the one-clock restriction).

Many problems remain open in this area, which already offers a wide range of problems to work on. An important problem is that of optimal reachability under lower-bound constraints in the general case, both in the case of weighted timed automata and weighted timed games. These are very natural problems I plan to work on in a very near future.

The extension to richer dynamics is another challenging direction. Most of the currently existing results involve linear observers, with the notable exception of our results on exponential observers in [BFLM10]. Extending the framework to different kinds of dynamics would evidently be of interest. One possible direction is the development of a general theory of weighted timed automata, similar to the theory of weighted automata [DKV09].

Finally, in order to circumvent undecidability of optimization problems, one possible direction is to rely on approximation techniques. The range of techniques is quite large in this direction, and several such techniques already exist for hybrid automata. Evaluating

the *quality* of the approximation would be especially interesting. This challenging problem could also be studied in conjunction with more realistic (hence less precise) models, as explained below.

Imprecise semantics of timed automata. Timed automata are governed with a mathematical semantics, which is convenient when applying formal methods but is not realistic. In particular, properties that have been ascertained in the mathematical setting, assuming arbitrary precision and speed of the hardware, might be lost when implementing the system on real, finite frequency hardware.

Several approaches have already been proposed to handle this problem (tube semantics, probabilistic semantics, parametrised sampled semantics, parametrised enlarged semantics, ...). While they all come with interesting and relevant alternative semantics for timed automata, I believe that there is more to come on this topic. This is one direction we are currently exploring with Patricia Bouyer-Decitre and Ocan Sankur. In particular, I think that extensions of *pseudometrics* on timed automata could be an interesting approach to this problem. Considering probabilistic imprecisions could also be relevant instead of the usual worst-case approach that has been considered so far.

The remark that undecidability proofs always require arbitrary precision in the model also opens a wide range of new problems. Some problems that are undecidable under the usual semantics of timed automata may reveal decidable under alternative, imprecise semantics. This is also a very exciting and promising direction I recently stepped into and would like to develop further.

The problem of imprecision also arises in the timed-game setting: besides imprecisions in the models, which may be dealt with using the approaches above, the synthesis of *implementable* strategies in timed games has not been tackled. In particular, strategies in timed games should (as much as possible) propose intervals of winning delays instead of just one delay. I would like to extend our first works on untimed *multi-strategies* to the timed setting, measuring how much precision is needed to play a strategy in a timed game.

Concurrent games. The recent extensions of ATL towards non-zero-sum objectives is something I would like to explore further. Being very recent, these approaches are still not precisely understood, and several problems remain open, such as the behavioural equivalence corresponding to ATL_{sc} or the analysis of the memory complexity of ATL_{sc} formulas.

The development of really concurrent timed games, such as timed CGSs, is also something I would like to work on. A first extension of this model, in order to make it powerful enough to encode timed game automata, is part of the short-term plans.

Finally, I will keep on working on the study of Nash equilibria (and other solution concepts) in timed (and possibly other infinite-state) games. Our recent results in this setting, computing equilibria in concurrent games using the new notion of *repellers*, are already very promising. Extending our study to mixed strategies is part of our future plans.

Bibliography

- [ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.
- [ACH94] Rajeev Alur, Costas Courcoubetis, and Thomas A. Henzinger. The observational power of clocks. In CONCUR'94, LNCS 836, p. 162–177. Springer, August 1994.
- [ACH97] Rajeev Alur, Costas Courcoubetis, and Thomas A. Henzinger. Computing accumulated delays in real time systems. *Formal Methods in System Design*, 11(2):137–155, August 1997.
- [ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In HSCC'92, LNCS 736, p. 209–229. Springer, 1993.
- [AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In ICALP'90, LNCS 443, p. 322–335. Springer, July 1990.
- [AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.
- [AFF⁺02] Roy Armoni, Limor Fix, Alon Flaisher, Rob Gerth, Boris Ginsburg, Tomer Kanza, Avner Landver, Sela Mador-Haim, Eli Singerman, Andreas Tiemeyer, Moshe Y. Vardi, and Yael Zbar. The ForSpec temporal logic: A new temporal property-specification language. In TACAS'02, LNCS 2280, p. 296–311. Springer, April 2002.
- [AFH94] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. In CAV'94, LNCS 818, p. 1–13. Springer, June 1994.
- [AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, January 1996.
- [ÅGJ07] Thomas Ågotnes, Valentin Goranko, and Wojciech Jamroga. Alternating-time temporal logics with irrevocable strategies. In TARK'07, p. 15–24, June 2007.

- [ÅGJ08] Thomas Ågotnes, Valentin Goranko, and Wojciech Jamroga. Strategic commitment and release in logics for multi-agent systems (extended abstract). Technical Report 08-01, Clausthal University of Technology, Germany, 2008.
- [AH89] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. In FOCS'89, p. 164–169. IEEE Comp. Soc. Press, October 1989.
- [AH92a] Rajeev Alur and Thomas A. Henzinger. Back to the future: Towards a theory of timed regular languages. In FOCS'92, p. 177–186. IEEE Comp. Soc. Press, October 1992.
- [AH92b] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In REX'91, LNCS 600, p. 74–106. Springer, 1992.
- [AH93] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, May 1993.
- [AH94] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–203, January 1994.
- [AHK97] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. In FOCS'97, p. 100–109. IEEE Comp. Soc. Press, October 1997.
- [AHK98] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. In COMPOS'97, LNCS 1536, p. 23–60. Springer, 1998.
- [AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, September 2002.
- [AHKV98] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In CONCUR'98, LNCS 1466, p. 163–178. Springer, September 1998.
- [AJ96] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, June 1996.
- [ALNR10] Natasha Alechina, Brian Logan, Nguyen Hoang Nga, and Abdur Rakib. Resource-bounded alternating-time temporal logic. In AAMAS'10, p. 481–488. International Foundation for Autonomous Agents and Multiagent Systems, May 2010.
- [ALP01] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In HSCC'01, LNCS 2034, p. 49–62. Springer, March 2001.
- [AMP95] Eugene Asarin, Oded Maler, and Amir Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138(1):35–65, 1995.

- [AMPS98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In SSC'98, p. 469–474. Elsevier Science, July 1998.
- [ASY07] Eugene Asarin, Gerardo Schneider, and Sergio Yovine. Algorithmic analysis of polygonal hybrid systems, part I: Reachability. *Theoretical Computer Science*, 379(1-2):231–265, June 2007.
- [BBB⁺08] Christel Baier, Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Marcus Größer. Almost-sure model checking of infinite paths in one-clock timed automata. In LICS'08. IEEE Comp. Soc. Press, June 2008.
- [BBBM08] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. **Quantitative model-checking of one-clock timed automata under probabilistic semantics**. In QEST'08, p. 55–64, Saint Malo, France, September 2008. IEEE Computer Society Press.
- [BBBR07] Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, October 2007.
- [BBFK06] Tomáš Brázdil, Václav Brožek, Vojtěch Forejt, and Antonín Kučera. Stochastic games with branching-time winning objectives. In LICS'06, p. 349–358. IEEE Comp. Soc. Press, July 2006.
- [BBGK07] Christel Baier, Tomáš Brázdil, Marcus Größer, and Antonín Kučera. Stochastic game logic. In QEST'07, p. 227–236. IEEE Comp. Soc. Press, September 2007.
- [BBL08] Patricia Bouyer, Ed Brinksma, and Kim Gulstrand Larsen. Staying alive as cheaply as possible. *Formal Methods in System Design*, 32(1):2–23, February 2008.
- [BBM06] Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. **Improved undecidability results on weighted timed automata**. *Information Processing Letters*, 98(5):188–194, June 2006.
- [BBM10a] Patricia Bouyer, Romain Brenguier, and Nicolas Markey. **Computing equilibria in two-player timed games via turn-based finite games**. In FORMATS'10, LNCS 6246, p. 62–76, Vienna, Austria, September 2010. Springer.
- [BBM10b] Patricia Bouyer, Romain Brenguier, and Nicolas Markey. **Nash equilibria for reachability objectives in multi-player timed games**. In CONCUR'10, LNCS 6269, p. 192–206, Paris, France, August-September 2010. Springer.
- [BBR05] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In FORMATS'05, LNCS 3829, p. 49–64. Springer, September 2005.

- [BC10] Luboš Brim and Jakub Chaloupka. Using strategy improvement to stay alive. Technical Report FIMU-RS-2010-03, Faculty of Informatics, Masaryk University, Brno, Czech Republic, March 2010.
- [BCFL04] Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim Gulstrand Larsen. Optimal strategies in priced timed game automata. In FSTTCS'04, LNCS 3328, p. 148–160. Springer, December 2004.
- [BCG88] Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1-2):115–131, July 1988.
- [BCM05] Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. **On the expressiveness of TPTL and MTL**. In FSTTCS'05, LNCS 3821, p. 432–443, Hyderabad, India, December 2005. Springer.
- [BCM10] Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. **On the expressiveness of TPTL and MTL**. *Information and Computation*, 208(2):97–116, February 2010.
- [BDL05] Gerd Behrmann, Alexandre David, and Kim Gulstrand Larsen. A tutorial on Uppaal, 2005.
- [BDLM09] Thomas Brihaye, Arnaud Da Costa, François Laroussinie, and Nicolas Markey. **ATL with strategy contexts and bounded memory**. In LFCS'09, LNCS 5407, p. 92–106, Deerfield Beach, FL, USA, January 2009. Springer.
- [BDMR09] Patricia Bouyer, Marie Duflot, Nicolas Markey, and Gabriel Renault. **Measuring permissivity in finite games**. In CONCUR'09, LNCS 5710, p. 196–210, Bologna, Italy, September 2009. Springer.
- [BFH⁺01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Gulstrand Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. In HSCC'01, LNCS 2034, p. 147–161. Springer, March 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 FORMATS'08, LNCS 5215, p. 33–47, Saint-Malo, France, September 2008. Springer.
- [BFLM10] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. **Timed automata with observers under energy constraints**. In HSCC'10, p. 61–70, Stockholm, Sweden, April 2010. ACM Press.
- [BGK⁺96] Johan Bengtsson, W. O. David Griffioen, Kåre J. Kristoffersen, Kim Gulstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Verification of an audio protocol with bus collision using UPPAAL. In CAV'96, LNCS 1102, p. 244–256. Springer, July-August 1996.

- [BGMR08] Thomas Brihaye, Mohamed Ghannem, Nicolas Markey, and Lionel Rieg. **Good friends are hard to find!** In TIME'08, p. 32–40, Montréal, Canada, June 2008. IEEE Computer Society Press.
- [BJ10] Nils Bulling and Wojciech Jamroga. Model checking agents with memory is harder than it seemed. In AAMAS'10, p. 633–640. International Foundation for Autonomous Agents and Multiagent Systems, May 2010.
- [BJL⁺10] Thomas Brihaye, Marc Jungers, Samson Lasaulce, Nicolas Markey, and Ghassan Oreiby. **Using model checking for analyzing distributed power control problems.** *EURASIP Journal on Wireless Communications and Networking*, 2010(861472), June 2010.
- [BLM07] Patricia Bouyer, Kim G. Larsen, and Nicolas Markey. **Model-checking one-clock priced timed automata.** In FoSSaCS'07, LNCS 4423, p. 108–122, Braga, Portugal, March 2007. Springer.
- [BLM08] Patricia Bouyer, Kim G. Larsen, and Nicolas Markey. **Model checking one-clock priced timed automata.** *Logical Methods in Computer Science*, 4(2:9), June 2008.
- [BLMO07] Thomas Brihaye, François Laroussinie, Nicolas Markey, and Ghassan Oreiby. **Timed concurrent game structures.** In CONCUR'07, LNCS 4703, p. 445–459, Lisbon, Portugal, September 2007. Springer.
- [BLMR06] Patricia Bouyer, Kim G. Larsen, Nicolas Markey, and Jacob Illum Rasmussen. **Almost optimal strategies in one-clock priced timed automata.** In FSTTCS'06, LNCS 4337, p. 345–356, Kolkata, India, December 2006. Springer.
- [BM07] Patricia Bouyer and Nicolas Markey. **Costs are expensive!** In FORMATS'07, LNCS 4763, p. 53–68, Salzburg, Austria, October 2007. Springer.
- [BMO⁺08] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, Philippe Schnoebelen, and James Worrell. **On termination for faulty channel machines.** In STACS'08, LIPIcs 1, p. 121–132, Bordeaux, France, February 2008. Leibniz-Zentrum für Informatik.
- [BMOW07] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. **The cost of punctuality.** In LICS'07, p. 109–118, Wrocław, Poland, July 2007. IEEE Computer Society Press.
- [BMOW08] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. **On expressiveness and complexity in real-time model checking.** In ICALP'08, LNCS 5126, p. 124–135, Reykjavik, Iceland, July 2008. Springer.
- [BMR06] Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. **Robust model-checking of linear-time properties in timed automata.** In LATIN'06, LNCS 3887, p. 238–249, Valdivia, Chile, March 2006. Springer.

- [BMR08] Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. *Robust analysis of timed automata via channel machines*. In FoSSaCS'08, LNCS 4962, p. 157–171, Budapest, Hungary, March-April 2008. Springer.
- [Bus87] Samuel R. Buss. The boolean formula value problem is in ALOGTIME. In STOC'87, p. 123–131. ACM Press, May 1987.
- [CD10] Krishnendu Chatterjee and Laurent Doyen. Energy parity games. In ICALP'10, LNCS 6199, p. 599–610. Springer, July 2010.
- [CdAHS03] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In EMSOFT'03, LNCS 2855, p. 117–133. Springer, October 2003.
- [CDF⁺05] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Gulstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In CONCUR'05, LNCS 3653, p. 66–80. Springer, August 2005.
- [CDHR10] Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Generalized mean-payoff and energy games. In FSTTCS'10, LIPIcs 8. Leibniz-Zentrum für Informatik, December 2010.
- [CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In LOP'81, LNCS 131, p. 52–71. Springer, 1982.
- [Čer93] Kārlis Čerāns. Decidability of bisimulation equivalences for parallel timer processes. In CAV'92, LNCS 663, p. 302–315. Springer, 1993.
- [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*, 8(2):244–263, April 1986.
- [CHP07] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. In CONCUR'07, LNCS 4703, p. 59–73. Springer, September 2007.
- [CY92] Costas Courcoubetis and Mihalis Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4):385–415, December 1992.
- [dAFH⁺03] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In CONCUR'03, LNCS 2761, p. 142–156. Springer, August-September 2003.
- [dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In FSE'01, p. 109–120. ACM Press, September 2001.
- [DDG⁺10] Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Toruńczyk. Energy and mean-payoff games with imperfect information. In CSL'10, LNCS 6247, p. 260–274. Springer, August 2010.

- [DDMR04] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. **Robustness and implementability of timed automata**. In FOR-MATS’04/FTRTFT’04, LNCS 3253, p. 118–133, Grenoble, France, September 2004. Springer.
- [DDMR08] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. **Robust safety of timed automata**. *Formal Methods in System Design*, 33(1-3):45–84, December 2008.
- [DDR04] Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics: From timed models to timed implementations. In HSCC’04, LNCS 2993, p. 296–310. Springer, March-April 2004.
- [DGR09] Laurent Doyen, Raffaella Gentilini, and Jean-François Raskin. Faster pseudo-polynomial algorithms for mean-payoff games. Technical Report 2009-121, CFV, 2009.
- [DKV09] Manfred Droste, Werner Kuich, and Walter Vogler. *Handbook of Weighted Automata*. Springer, 2009.
- [DLM10] Arnaud Da Costa, François Laroussinie, and Nicolas Markey. **ATL with strategy contexts: Expressiveness and model checking**. In FSTTCS’10, LIPIcs 8, p. 120–132, Chennai, India, December 2010. Leibniz-Zentrum für Informatik.
- [EH83] E. Allen Emerson and Joseph Y. Halpern. ”sometimes” and ”not never” revisited: On branching versus linear time. In POPL’83, p. 127–140. ACM Press, January 1983.
- [EH86] E. Allen Emerson and Joseph Y. Halpern. ”sometimes” and ”not never” revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, January 1986.
- [EJ91] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In FOCS’91, p. 368–377. IEEE Comp. Soc. Press, October 1991.
- [EJ99] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, September 1999.
- [EM79] Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, June 1979.
- [EMSS92] E. Allen Emerson, Aloysius Ka-Lau Mok, A. Prasad Sistla, and Jayashankar Srinivasan. Quantitative temporal reasoning. *Real-Time Systems*, 4:331–352, 1992.

- [EY05] Kousha Etessami and Mihalis Yannakakis. Recursive Markov decision processes and recursive stochastic games. In ICALP'05, LNCS 3580, p. 891–903. Springer, July 2005.
- [EY07] Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points (extended abstract). In FOCS'07, p. 113–123. IEEE Comp. Soc. Press, October 2007.
- [FH10] Nathanaël Fijalkow and Florian Horn. The surprising complexity of generalized reachability games, 2010.
- [FR07] Carlo A. Furia and Matteo Rossi. On the expressiveness of MTL variants over dense time. In FORMATS'07, LNCS 4763, p. 163–178. Springer, October 2007.
- [Fre06] Tim French. *Bisimulation Quantifiers for Modal Logics*. Phd thesis, School of Computer Science & Software Engineering, University of Western Australia, Australia, December 2006.
- [FS01] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, April 2001.
- [GHJ97] Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan. Robust timed automata. In HART'97, LNCS 1201, p. 331–345. Springer, March 1997.
- [GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.
- [GPSS80] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In POPL'80, p. 163–173. ACM Press, January 1980.
- [GvD06] Valentin Goranko and Govert van Drimmelen. Complete axiomatization and decidability of alternating-time temporal logic. *Theoretical Computer Science*, 353(1-3):93–117, March 2006.
- [HBM⁺10] Paul Hunter, Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. **Computing rational radical sums in uniform TC⁰**. In FSTTCS'10, LIPIcs 8, p. 308–316, Chennai, India, December 2010. Leibniz-Zentrum für Informatik.
- [Hen96] Thomas A. Henzinger. The theory of hybrid automata. In LICS'96, p. 278–292. IEEE Comp. Soc. Press, July 1996.
- [Hen98] Thomas A. Henzinger. It's about time: Real-time logics reviewed. In CONCUR'98, LNCS 1466, p. 439–454. Springer, September 1998.
- [HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What is decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, August 1998.

- [HM85] Matthew C. B. Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1), January 1985.
- [HMP05] Thomas A. Henzinger, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying similarities between timed systems. In FORMATS'05, LNCS 3829, p. 226–241. Springer, September 2005.
- [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, June 1994.
- [Hoa80] Charles Antony Richard Hoare. A model for communicating sequential processes. In *On the Construction of Programs – An advanced course*, p. 229–254. Cambridge University Press, 1980.
- [HP06] Thomas A. Henzinger and Vinayak S. Prabhu. Timed alternating-time temporal logic. In FORMATS'06, LNCS 4202, p. 1–17. Springer, September 2006.
- [HR99] Yoram Hirshfeld and Alexander Rabinovich. Quantitative temporal logic. In CSL'99, LNCS 1862, p. 172–187. Springer, September 1999.
- [HR05] Yoram Hirshfeld and Alexander Rabinovich. Timer formulas and decidable metric temporal logic. *Information and Computation*, 198(2):148–178, May 2005.
- [HRS98] Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In ICALP'98, LNCS 1443, p. 580–591. Springer, July 1998.
- [HSSL97] Klaus Havelund, Arne Skou, Kim Gulstrand Larsen, and Kristian Lund. Formal modelling and analysis of an audio/video protocol: An industrial case study using Uppaal. In RTSS'97, p. 2–13. IEEE Comp. Soc. Press, December 1997.
- [JD08] Wojciech Jamroga and Jürgen Dix. Model checking abilities of agents: A closer look. *Theory of Computing Systems*, 42(3):366–410, April 2008.
- [Kam68] Hans W. Kamp. *Tense Logic and the Theory of Linear Order*. Phd thesis, Computer Science Department, University of California at Los Angeles, USA, 1968.
- [KF09] Lars Kuhtz and Bernd Finkbeiner. LTL path checking is efficiently parallelizable. In ICALP'09, LNCS 5557, p. 235–246. Springer, July 2009.
- [Koy87] Ron Koymans. Specifying message passing and real-time systems with real-time temporal logic. In ESPRIT'87. Elsevier Science, September 1987.
- [Koz83] Dexter C. Kozen. Results on the propositional μ -calculus. *Theoretical Computer Science*, 27:333–354, 1983.

- [KP95] Orna Kupferman and Amir Pnueli. *Once and for all*. In LICS'95, p. 25–35. IEEE Comp. Soc. Press, June 1995.
- [KP04] Magdalena Kacprzak and Wojciech Penczek. Unbounded model checking for alternating-time temporal logic. In AAMAS'04, p. 646–653. IEEE Comp. Soc. Press, August 2004.
- [KPSY99] Yonit Kesten, Amir Pnueli, Joseph Sifakis, and Sergio Yovine. Decidable integration graphs. *Information and Computation*, 150(2):209–243, May 1999.
- [KS83] Paris C. Kanellakis and Scott A. Smolka. CCS expressions finite state processes, and three problems of equivalence. In PODC'83, p. 228–240. ACM Press, August 1983.
- [Kup99] Orna Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. *Journal of Logic and Computation*, 9(2):135–147, April 1999.
- [KVW00] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model-checking. *Journal of the ACM*, 47(2):312–360, March 2000.
- [KVW01] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. Module checking. *Information and Computation*, 164(2):322–344, January 2001.
- [Lan07] Martin Lange. Linear-time logics around PSL: Complexity, expressiveness, and a little bit of succinctness. In CONCUR'07, LNCS 4703, p. 90–104. Springer, September 2007.
- [LMO06] François Laroussinie, Nicolas Markey, and Ghassan Oreiby. **Model checking timed ATL for durational concurrent game structures**. In FORMATS'06, LNCS 4202, p. 245–259, Paris, France, September 2006. Springer.
- [LMO07] François Laroussinie, Nicolas Markey, and Ghassan Oreiby. **On the expressiveness and complexity of ATL**. In FoSSaCS'07, LNCS 4423, p. 243–257, Braga, Portugal, March 2007. Springer.
- [LMO08] François Laroussinie, Nicolas Markey, and Ghassan Oreiby. **On the expressiveness and complexity of ATL**. *Logical Methods in Computer Science*, 4(2:7), May 2008.
- [LMS02a] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. **On model checking durational Kripke structures (extended abstract)**. In FoSSaCS'02, LNCS 2303, p. 264–279, Grenoble, France, April 2002. Springer.
- [LMS02b] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. **Temporal logic with forgettable past**. In LICS'02, p. 383–392, Copenhagen, Denmark, July 2002. IEEE Computer Society Press.

- [LMS04] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. **Model checking timed automata with one or two clocks**. In CONCUR'04, LNCS 3170, p. 387–401, London, UK, August 2004. Springer.
- [LMS06] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. **Efficient timed model checking for discrete-time systems**. *Theoretical Computer Science*, 353(1-3):249–271, March 2006.
- [LN00] Salvatore La Torre and Margherita Napoli. A decidable dense branching-time temporal logic. In FSTTCS'00, LNCS 1974, p. 139–150. Springer, December 2000.
- [LN01] Salvatore La Torre and Margherita Napoli. Timed tree automata with an application to temporal logic. *Acta Informatica*, 38(2):89–116, 2001.
- [LP07] Yuri M. Lifshits and Dmitri S. Pavlov. Potential theory for mean payoff games. *Journal of Mathematical Science*, 145(3):4967–4974, September 2007.
- [LR08] Kim Gulstrand Larsen and Jacob Illum Rasmussen. Optimal conditional reachability for multi-priced timed automata. *Theoretical Computer Science*, 390(2-3):197–213, January 2008.
- [LS94] François Laroussinie and Philippe Schnoebelen. A hierarchy of temporal logics with past. In STACS'94, LNCS 775, p. 47–58. Springer, February 1994.
- [LY94] Kim Gulstrand Larsen and Wang Yi. Time abstracted bisimulation: Implicit specifications and decidability. In MFPS'93, LNCS 802, p. 160–176. Springer, 1994.
- [Mar75] Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, September 1975.
- [MH84] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω -words. *Theoretical Computer Science*, 32:321–330, 1984.
- [Mil80] Robin Milner. *A Calculus of Communicating Systems*, LNCS 92. Springer, 1980.
- [MMV10] Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. Reasoning about strategies. In FSTTCS'10, LIPIcs 8, p. 133–144. Leibniz-Zentrum für Informatik, December 2010.
- [MNP06] Oded Maler, Dejan Nickovic, and Amir Pnueli. From MITL to timed automata. In FORMATS'06, LNCS 4202, p. 274–289. Springer, September 2006.
- [MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In STACS'95, LNCS 900, p. 229–242. Springer, March 1995.

- [MR04] Nicolas Markey and Jean-François Raskin. [Model checking restricted sets of timed paths](#). In CONCUR'04, LNCS 3170, p. 432–447, London, UK, August 2004. Springer.
- [MR06] Nicolas Markey and Jean-François Raskin. [Model checking restricted sets of timed paths](#). *Theoretical Computer Science*, 358(2-3):273–292, August 2006.
- [MS85] David E. Muller and Paul E. Schupp. Alternating automata on infinite objects, determinacy and Rabin's theorem. In EPIT'84, LNCS 192, p. 99–107. Springer, 1985.
- [MS87] David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54(2-3):267–276, October 1987.
- [MS95] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1-2):69–107, April 1995.
- [MS03] Nicolas Markey and Philippe Schnoebelen. [Model checking a path \(preliminary report\)](#). In CONCUR'03, LNCS 2761, p. 251–265, Marseilles, France, August 2003. Springer.
- [MS04a] Nicolas Markey and Philippe Schnoebelen. [A PTIME-complete matching problem for SLP-compressed words](#). *Information Processing Letters*, 90(1):3–6, January 2004.
- [MS04b] Nicolas Markey and Philippe Schnoebelen. [Symbolic model checking for simply-timed systems](#). In FORMATS'04/FTRTFT'04, LNCS 3253, p. 102–117, Grenoble, France, September 2004. Springer.
- [MS04c] Nicolas Markey and Philippe Schnoebelen. [TSMV: A symbolic model checker for quantitative analysis of systems](#). In QEST'04, p. 330–331, Enschede, The Netherlands, September 2004. IEEE Computer Society Press.
- [MS06] Nicolas Markey and Philippe Schnoebelen. [Mu-calculus path checking](#). *Information Processing Letters*, 97(6):225–230, March 2006.
- [Ore08] Ghassan Oreiby. *Logiques temporelles pour le contrôle temporisé*. Thèse de doctorat, Lab. Spécification & Vérification, ENS Cachan, France, December 2008.
- [OW05] Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In LICS'05, p. 188–197. IEEE Comp. Soc. Press, July 2005.
- [OW06a] Joël Ouaknine and James Worrell. On metric temporal logic and faulty Turing machines. In FoSSaCS'06, LNCS 3921, p. 217–230. Springer, March 2006.

- [OW06b] Joël Ouaknine and James Worrell. Safety metric temporal logic is fully decidable. In TACAS'06, LNCS 3920, p. 411–425. Springer, March 2006.
- [Pin07] Sophie Pinchinat. A generic constructive solution for concurrent games with expressive constraints on strategies. In ATVA'07, LNCS 4762, p. 253–267. Springer, October 2007.
- [Pnu77] Amir Pnueli. The temporal logic of programs. In FOCS'77, p. 46–57. IEEE Comp. Soc. Press, October–November 1977.
- [Pri67] Arthur N. Prior. *Past, Present and Future*. Oxford University Press, 1967.
- [Pri68] Arthur N. Prior. *Papers on Time and Tense*. Oxford University Press, 1968.
- [Pur98] Anuj Puri. Dynamical properties of timed automata. In FTRTFT'98, LNCS 1486, p. 210–227. Springer, September 1998.
- [QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In SOP'82, LNCS 137, p. 337–351. Springer, April 1982.
- [Ras99] Jean-François Raskin. *Logics, Automata and Classical theories for Deciding Real Time*. Thèse de doctorat, FUNDP, Namur, Belgium, June 1999.
- [Rey10] Mark Reynolds. The complexity of temporal logic over the reals. *Annals of Pure and Applied Logic*, 161(8):1063–1096, May 2010.
- [RS97] Jean-François Raskin and Pierre-Yves Schobbens. State clock logic: A decidable real-time logic. In HART'97, LNCS 1201, p. 33–47. Springer, March 1997.
- [RS99] Jean-François Raskin and Pierre-Yves Schobbens. The logic of event-clocks: Decidability, complexity and expressiveness. *Journal of Automata, Languages and Combinatorics*, 4(3):247–286, 1999.
- [Rut11] Michał Rutkowski. Two-player reachability-price games on single-clock timed automata. In QAPL'11, EPTCS, April 2011.
- [SC85] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, July 1985.
- [Sch08] Sven Schewe. ATL* satisfiability is 2EXPTIME-complete. In ICALP'08, LNCS 5126, p. 373–385. Springer, July 2008.
- [SJ01] Zdeněk Sawa and Petr Jančar. P-hardness of equivalence testing on finite state processes. In SOFSEM'01, LNCS 2234, p. 326–335. Springer, November 2001.
- [Sto76] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

- [SVW87] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with applications to temporal logics. *Theoretical Computer Science*, 49:217–237, 1987.
- [TA99] Stavros Tripakis and Karine Altisen. On-the-fly controller synthesis for discrete and dense-time systems. In FM’99, LNCS 1708, p. 233–252. Springer, September 1999.
- [TFL10] Claus Thrane, Uli Fahrenberg, and Kim Gulstrand Larsen. Quantitative analysis of weighted transition systems. *Journal of Logic and Algebraic Programming*, 79(7):689–703, October 2010.
- [Var88] Moshe Y. Vardi. A temporal fixpoint calculus. In POPL’88, p. 250–259. ACM Press, January 1988.
- [Var94] Moshe Y. Vardi. Nontraditional applications of automata theory. In TACS’94, LNCS 789, p. 575–597. Springer, April 1994.
- [Var95] Moshe Y. Vardi. Alternating automata and program verification. In *Computer Science Today: Recent Trends and Developments*, LNCS 1000, p. 471–485. Springer, 1995.
- [vD03] Govert van Drimmelen. Satisfiability in alternating-time temporal logic. In LICS’03, p. 208–217. IEEE Comp. Soc. Press, June 2003.
- [Vol99] Heribert Vollmer. *Introduction to circuit complexity*. Springer, 1999.
- [VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In LICS’86, p. 332–344. IEEE Comp. Soc. Press, June 1986.
- [WLWW06] Dirk Walther, Carsten Lutz, Frank Wolter, and Michael Wooldridge. ATL satisfiability is indeed EXPTIME-complete. *Journal of Logic and Computation*, 16(6):765–787, December 2006.
- [Wol83] Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1-2):72–99, 1983.
- [WvdHW07] Dirk Walther, Wiebe van der Hoek, and Michael Wooldridge. Alternating-time temporal logic with explicit strategies. In TARK’07, p. 269–278, June 2007.
- [Yov97] Sergio Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2):123–133, October 1997.
- [ZP96] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, May 1996.