UMONS
Faculté des Sciences
Département de Mathématiques

Verification and synthesis of MITL through alternating timed automata

Morgane Estiévenart

A dissertation submitted in fulfilment of the requirements of the degree of *Docteur en Sciences*

Advisor
Pr. Thomas Brihaye
Université de Mons, Belgium
Pr. Gilles Geeraerts
Université libre de Bruxelles, Belgium

Jury
Pr. Thomas Brihaye
Université de Mons, Belgium
Pr. Véronique Bruyère
Université de Mons, Belgium
Pr. Gilles Geeraerts
Université libre de Bruxelles, Belgium
Pr. Nicolas Markey
École Normale Supérieure de Cachan, France
Pr. Jean-François Raskin
Université libre de Bruxelles, Belgium
Pr. Cédric Rivière
Université de Mons, Belgium

September 2015

Acknowledgements

Mes premiers remerciements vont à mes deux co-directeurs de thèse Thomas Brihaye et Gilles Geeraerts, qui ont cru en moi en m'acceptant comme doctorante. Je voudrais leur exprimer ma gratitude pour m'avoir accompagnée tout au long de ces quatre années et pour m'avoir guidée dans chacune des activités de la vie d'un chercheur que j'ai découverte à leurs côtés. Votre expérience et vos conseils judicieux ont été d'une grande aide: ils m'ont notamment permis de me familiariser avec la formalisation d'idées clés et la rédaction (en anglais !) de papiers. Merci également pour votre compréhension face à mes difficultés en anglais et mon aversion des longs voyages avec lesquelles il n'a pas toujours été facile de composer.

Je remercie ensuite tous les membres de mon jury pour avoir eu le courage d'accepter la relecture de ma thèse et la tâche de la juger.

Je me dois aussi de remercier ma famille qui a toujours cru en moi, sûrement plus que je n'en suis moi-même capable. Merci d'avoir toujours été là pour moi et de m'avoir toujours soutenue dans mes choix. Merci de m'aider et de m'épauler dans mes entreprises, même les plus farfelues ! Merci pour votre gentillesse et vos attentions, biens rares dont j'ai la chance de pouvoir profiter.

Je n'oublie pas non plus mes amis, pour les moments de détente nécessaires à la décompression. Je les remercie pour tous ces moments partagés, ces nombreux

fous rires et les soirées inoubliables au Kot et Jeux ou ailleurs, souvent autour de jeux de société, toujours dans la joie de vivre et la bonne humeur !

Table of Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Nowadays, computer systems are frequently used in our daily life. We find them in a multitude of current life devices : smartphones, cars, online banking, ... Such systems often work in continuous interaction with their environment and are so called *reactive systems*.

For economic reasons as well as security purposes, it is important that reactive systems do not experience any error. In certain fields, this property is even essential. For instance, it is unacceptable that an air traffic control system is erroneous: numerous lifes are at stake. However, by the past, several reactive systems caused casualties. We can cite the death by radiation overdose of at least six cancer patients, due to a software bug in the control part of the radiation therapy machine Therac-25, between 1985 and 1987. The correctness of reactive systems may also cause big financial losses. It was the case for Intel in 1994: they commercialized processors 'Intel Pentium II' containing a floating point division bug. It cost $475 million to replace these erroneous processors. Another well-known bug example is the explosion of the rocket Ariane 5 in 1996, a few seconds after its launch. This was due to a conversion in its guidance system, of a 64-bit

floating point number to a 16-bit signed integer.

To avoid those disastrous situations, the need of techniques and tools to verify the correctness of reactive systems grew. The classical technique used to try to ensure the correctness of a software is the *testing*. However, to test a system for correctness takes a lot of time and, furthermore, it is not an infallible technique. Indeed, the testing of a software enables to verify several (various) behaviours of the system... but not all of them! So, as Dijkstra said, testing enables to detect the presence of a bug, but not to ensure their absence. The field of *formal verification*, a branch of theoretical computer science, is dedicated to the research of solutions for these problems. The most famous approach to this problem is probably the *model-checking* technique [10].

Model-checking. Model-checking is a formal verification method that enables to prove the correctness of reactive systems. To perform model-checking, a *model-checker* is given two entries: (i) a representation of the system to analyse and (ii) a property the system must satisfy. Generally, the system is modeled by means of a *Büchi automaton* $\mathcal{B}$ [10, 37] and the property to prove is usually expressed using a *linear temporal logic* (LTL) formula [53]. Establishing the correctness of the system amounts to showing that the language $L(\mathcal{B})$ of the system (representing all its possible behaviors) is included in the language of a Büchi automaton $\mathcal{B}_\Phi$ recognizing the language $[\![\Phi]\!]^\omega$ of the formula (representing the *correct* behaviors). More recently, another technique uses a preliminary translation of Φ into an *alternating automaton* $\mathcal{A}_\Phi$ recognizing $[\![\Phi]\!]^\omega$. $\mathcal{A}_\Phi$ is then turned into a Büchi automaton $\mathcal{B}_\Phi$ recognizing the same language. This technique enables to construct $\mathcal{B}_\Phi$ from $\mathcal{A}_\Phi$ *on the fly*, so that an answer to the model-checking problem might be given back before the entire automaton $\mathcal{B}_\Phi$ is constructed.
Those checking methods have reached a deep maturity level, as can be seen from the various industrial and academical existing tools, such as SPIN [58] and nuSMV [24]. Recently, a new promising approach, based on *antichains* has been developed in the academical sector. This approach has allowed to revisit several classical algorithms as language inclusion and universality for automata over finite

words [64] and infinite words (with Büchi acceptance condition) [29]. Antichains also enable to revisit algorithms for the LTL satisfiability and model-checking problems [66]. Promising software tools, as ALASKA [65] and ALPAGA [13], based on those new algorithms, have allowed to quantify the efficiency of this antichain approach.

Although those tools, used in the industrial sector, enabled to verify a multitude of reactive systems, the verification of a lot of them is limited by the expressivity of LTL. Indeed, LTL only allows to express *qualitative properties* about the sequence of events happening in the systems. This logic allows, for example, to express the following property: 'every request will finally be followed by a grant'. In reactive system modeling, it is often necessary to express real-time *quantitative* constraints. In the previous example, the guaranty of the existence of a grant might not be a sufficient information, in some practical cases. As a matter of fact, the elapsed time between the request and the grant cannot be ignored to ensure the correctness of the system. LTL does not allow to express those quantitative aspects, therefore it is natural to consider real-time extensions of this logic as the Metric Temporal Logic (MTL), introduced in [39]. MTL allows to express, for example, the following property: 'Every request will be followed by a grant within three seconds'. *Timed automata* [4], taking into account those real-time aspects present in the systems, have also been introduced. Those kind of automata thus allow to describe reactive systems taking into account their real-time aspects.

Hence, timed automata and MTL compose a 'real-time' twin of Büchi automata and LTL. Unfortunately, the MTL model-checking problem over infinite words is undecidable [7] and it is decidable (but non primitive recursive) over finite words (for the *pointwise semantics*). The couple timed automata/MTL is therefore not a good candidate to reach real-time verification methods that could be used in practice. This explains why very few tools performing real-time verification are actually available, with the notable exception of the UPPAAL tool [41], that can only check a restricted set of non-quantitative properties (no real-time). Practical and efficient tools allowing to verify a wide range of real-time properties over models as automata would nevertheless be useful as well to the academic sector

as to the industrial one. It is in this spirit that several research works considered restrictions of MTL, to get round the undecidability limit. This thesis also takes part in this line of research.

Metric Interval Temporal Logic. Facing the undecidability and high complexity level over the problems linked to MTL, several fragments of MTL were investigated. A noticeable one is the Metric Interval Temporal Logic (MITL), a syntactic fragment of MTL [5]. Its model-checking was proved to be 2EXPSPACE-complete in [5], thanks to the construction, for any MITL formula Φ, of a Büchi timed automaton $\mathcal{B}_\Phi$ recognizing $[\![\Phi]\!]^\omega$. However, this construction is unintuitive and not amenable to an efficient implementation. Indeed, the definition of $\mathcal{B}_\Phi$ does not enable to construct this automaton *on the fly*, but obliges to construct it entirely before we can compare it to the language of the system of the model-checking problem. An alternative technique, based on the notion of *signal* has been proposed by Maler et al. in [44]. However the semantics used there slightly differs from that of [5]. As far as we know, these theoretical results never led to any implementation of an MITL model-checker. Notice that, as a twin of the LTL case, Ouaknine and Worrell [51] showed how each MTL formula Φ (and so, in particular, each MITL formula) can be translated into an *alternating timed automaton with one clock* (OCATA) recognizing $[\![\Phi]\!]$ (over finite words). However, this line of research had never been pursued, in way to perform MITL model-checking using this translation. This is the object of the first part of this thesis.

Reactive synthesis. As we have just seen, model-checking enables to automatically verify the correctness of a given system, regarding a desired property. Nevertheless, this verification can only be performed once the system has already been developed. This way, if a model-checker detects an error, the system must be design again. To avoid this 'trial and error' approach, the *reactive synthesis* has been proposed. Given a property, expressed by an LTL formula, the reactive synthesis problem consists in finding how a system might be designed to satisfy the given property. This way, the created system is correct by construction: it is

not necessary to verify it anymore.
The LTL reactive synthesis problem was first studied in [54] and [1]: it was proved to be 2EXPTIME-complete. This high complexity has first hindered the development of algorithm, efficient in practice, to solve the LTL reactive synthesis problem. More recently, the use of antichains and efficient data structures enabled to overcome those difficulties and to develop interesting heuristics to solve this problem [32].
As for the model-checking problem, real-time twins to the LTL reactive synthesis have been proposed. On the one hand, the reactive synthesis of the *Event-Clock Logic* (ECL) was investigated in [36], but was showed to be undecidable. On the second hand, a reactive synthesis using timed automata and MTL has also been investigated. In [31], the authors consider a specification represented by a timed automaton. They show that the reactive synthesis defined this way is undecidable. However, the problem becomes again decidable when fixing *a priori* the resources (number of given clocks, maximal constant used and the precision) provided to synthesize a correct system. In [17], the MTL reactive synthesis is considered. Once again, it is showed to be undecidable in general but decidable for fixed ressources, over finite words. Precisely, the MTL reactive synthesis, with fixed resources, over finite words, is showed to be a non-primitive recursive problem. This theoretical result hinders to develop an algorithm solving this problem and feasible in practice.

Structure and contributions of the thesis. Let us now present the contributions of each chapter of this thesis.

Chapter 2 is dedicated to basic definitions. We also present some existing results related to the framework of this thesis. In particular, we present the translation from each MTL formula Φ to an OCATA recognizing $[\![\Phi]\!]$ ([51]). The thesis is then divided into two parts, respectively dedicated to the model-checking and the reactive synthesis of MITL.

Part I. This part is dedicated to the MITL model-checking. In Chapter 3, we introduce a new semantics for OCATA, parametrized by an *approximation func-*

tion. We prove that, in general, the language it produces is an *underapproximation* of the language obtained with the classical semantics. Chapter 4 is devoted to the setting of finite words. We show that, when considering an OCATA $\mathcal{A}_\Phi$ recognizing the language of an MITL formula Φ and using a particular approximation function, our semantics and the classical one correspond, i.e. $\mathcal{A}_\Phi$ also recognizes $[\![\Phi]\!]$ with our semantics (parametrized by the appropriate approximation function). Using our new semantics, we then show that we can turn $\mathcal{A}_\Phi$ into a timed automaton over finite words recognizing $[\![\Phi]\!]$, in an intuitive way. Then, we present our techniques to perform *on the fly* MITL model-checking from $\mathcal{A}_\Phi$, using regions as well as zones. We furthermore introduce some heuristics using antichains for these algorithms. The results obtained in Chapters 3 and 4 were subject to a publication: [19] (arXiv reference: [20]). Chapter 5 is dedicated to infinite words: its aim is to extend the results of Chapter 4 to this setting. Although this extension may seem intuitive, the formal mecanism to obtain it is not trivial. In particular, we need to introduce and study a particular class of OCATA, that we called *tree-like OCATA* (TOCATA). We remark that each OCATA $\mathcal{A}_\Phi$ produced from an MITL formula Φ is in fact a TOCATA. A particularly interesting property of TOCATA is that they are easily complementable. This is the basis result enabling to extend the results obtained over finite words to the setting of infinite words. In Chapter 6, we present the results obtained using our prototype of tool implementing the algorithms of Chapters 4 and 5, on some benchamarks. The contributions obtained in Chapters 3 and 4 were also subject of a publication: [21] (arXiv reference: [22]).

Part II. In this second part, we study the MITL reactive synthesis with fixed resources. We follow the techniques developed in [17] over MTL, for the particular case of MITL. In this paper, an OCATA $\mathcal{A}_\Phi$ is used to represent the language $[\![\Phi]\!]$ (over finite words) of the MTL formula Φ of the reactive synthesis problem. We show how the techniques of [17] may be adapted using our new semantics over $\mathcal{A}_\Phi$, for an MITL formula Φ. Then, we show that, while the MTL reactive synthesis was proved to be a non primitive recursive problem in [17], we provide an algorithm executing in time triply exponential in the size of Φ for the particular

case of MITL.

We conclude this thesis by Chapter 9, in which we list some possible extensions of our work.

CHAPTER 2

Preliminaries

In this chapter, we recall some basic notions and fix useful notations. We will furthermore discuss about different kinds of automata and temporal logics. The first sections are addressed to untimed automata and untimed logics, we then move on to the timed setting. In the first sections, we will focus on the *pointwise semantics* of all these objects. Then, for the sake of completeness, we will dedicate a section to the *continuous semantics* for the automata and temporal logics previously defined. However, in the following chapters, we will only be interested in the pointwise semantics: the contributions of this thesis only concern this setting.

The first section fixes notations for *intervals* and recalls the notions of *words*, *timed words* and *language*.

The following sections concerns the untimed setting, over the pointwise semantics. We begin by recalling the notions of *automaton* (over finite words), of *Büchi automaton* (over infinite words), of *runs* and *languages*. We will then be interested in their *determinized* versions and in their *complements*. We finally

present well-known results about their *emptiness* and *universality* problems.
We also present the classes of *alternating automata* over finite and infinite words: we recall how are defined their *runs* and *languages*. We then present an existing method to translate an alternating automaton over finite words into an automaton over finite words recognizing the same language. We end exhibiting such a translation for the setting of infinite words.
The Linear Temporal Logic (LTL) is the first logic we are interested in. We recall its *syntax* and*pointwise semantics*. Several problems over LTL were studied in the litterature. We consider the LTL *satisfiability*, *model-checking* and *reactive synthesis* problems: we present the associated decidability and complexity results. We finally recall an interesting translation from LTL formulas to alternating automata recognizing the same languages.

We then enter the timed setting, over the pointwise semantics. We observe *timed automata* (over finite words) and *Büchi timed automata* (over infinite words), their *runs* and *languages*. We consider their *determinization* and *determinizability* problems, and also discuss their *complementation* and their *universality* problems.
The classes of *alternating timed automata with one clock* (OCATA), over finite and infinite words, are the last kinds of automata we are interested in. We recall the notions of *run* and *languages* of such automata. We then recall several well-known results about their *emptiness* problem. Finally, we discuss their *complementation*.
Then, we recall the definitions of the *Metric Temporal logic* (MTL), its *syntax* and its *pointwise semantics*. We consider, as twins of the main problems over LTL already defined, the MTL *satisfiability*, *model-checking* and *reactive synthesis* problems. We present several alternative definitions of the MTL reactive synthesis problem. We exhibit the known decidability and undecidability results about them. As well as there exists a translation from LTL to alternating automata, each MTL formula can be translated into an OCATA recognizing its language. This last translation will be described through a complete section and used all along this thesis.

A section is then addressed to the *continuous semantics*. We define *timed state sequences* that are twins of timed words and recall the MTL continuous semantics. We present those definitions for the sake of completeness but they will not be used anymore for the rest of this thesis.

We end this chapter displaying some alternative real-time logics: we present their *syntax* and their *pointwise semantics*. For the sake of completeness, we also present their *continuous semantics*, although the contributions of this thesis only concern the pointwise one. More precisely, we will consider the *Metric Interval Temporal Logic* (MITL), its fragment $\mathrm{MITL}_{0,\infty}$, the Event-Clock Logic (ECL) and its fragment $\mathrm{LTL}_{\lhd}$. We exhibit some well-known results about these logics. In particular, the MITL *satisfiability* and *model-checking* problems, to which we will adress a complete part of this thesis, are known to be EXPSPACE-complete.

2.1 Basic notions

Let $\mathbb{R}$, $\mathbb{R}^+$, $\mathbb{N}$ and $\mathbb{N}_0$ denote respectively the sets of real, non-negative real, natural and strictly positive natural numbers.

The notion of 'interval' will be often used is this thesis. It is a key point in the definitions of the real-time logics and automata we will use. Here is its formal definition and some useful notations.

Definition 2.1. *We call* interval *a convex subset of* $\mathbb{R}$. *We rely on the classical notation* $\langle a, b\rangle$ *for intervals, where* $\langle$ *and* $\rangle$ *are* $[$ *or* $]$, $a \in \mathbb{R} \cup \{-\infty\}$, $b \in \mathbb{R} \cup \{+\infty\}$ *and* $a \leqslant b$.
For an interval $I = \langle a, b\rangle$, *we call* a *and* b *the* endpoints *of* I. *We let* $\inf(I) = a$ *be the* infimum *of* I, $\sup(I) = b$ *be its* supremum *and* $|I| = \sup(I) - \inf(I)$ *be its* length. *If* $\langle$ *is* $[$ *(respectively* $]$*) we say that* I *is left-closed (respectively left-open). If* $\rangle$ *is* $]$ *(respectively* $[$*) we say that* I *is right-closed (respectively right-open).*

Definition 2.2. *We note* $\mathcal{I}(\mathbb{R})$ *the set of all intervals. Similarly, we note* $\mathcal{I}(\mathbb{R}^+)$

(respectively $\mathcal{I}(\mathbb{N}^\infty)$) the set of all intervals whose endpoints are in $\mathbb{R}^+$ (respectively in $\mathbb{N} \cup \{+\infty\}$).

Definition 2.3. *Let $I, J \in \mathcal{I}(\mathbb{R})$, $t \in \mathbb{R}^+$, $r \in \mathbb{R}$ and $\bowtie \in \{<, >\}$. we note:*

- *$I + t$ for the interval $\{i + t \in \mathbb{R} \mid i \in I\}$,*
- *$I - t$ for the interval $\{i - t \in \mathbb{R} \mid i \in I$ and $i \geqslant t\}$,*
- *$I < J$ iff $\forall i \in I, \forall j \in J : i < j$,*
- *$I \bowtie r$ iff $\forall i \in I, i \bowtie r$.*

Definition 2.4. *For $I_1, I_2 \in \mathbb{R}$, with $I_1 = \langle_1\ a, b\ \rangle_1$, $I_2 = \langle_2\ c, d\ \rangle_2$ and $I_1 < I_2$, we say that I_1 and I_2 are* adjacent *iff $b = c$ and $I_1 \cup I_2 = \langle_1\ a, d\ \rangle_2$.*

The logics defined later enable to provide properties about *words* and *timed words*. In the same way, the automata we will defined are acceptors of (timed) words. These words are defined over an *alphabet*, which is a (finite or infinite) set of elements called *letters*.

Definition 2.5. *A* word *over the alphabet Σ is a finite or infinite sequence $\overline{\sigma} = \sigma_1\sigma_2\sigma_3\ldots$ of elements in Σ.*
We denote by $|\overline{\sigma}| = n$ the length of $\overline{\sigma}$: it is the number of elements of $\overline{\sigma}$, when $\overline{\sigma}$ is finite; and $+\infty$ otherwise.
We denote by ϵ the empty word. $|\epsilon| = 0$.
We denote by $\Sigma^\star$ the set of all finite words over Σ and by Σ^ω the set of all infinite words over Σ.

Example 2.6. Let us consider the alphabet $\Sigma = \{a, b\}$.
$\overline{\sigma} = aaabab$ is an element of $\Sigma^\star$: it is a finite word over Σ. The length of $\overline{\sigma}$ is equal to 6: $|\overline{\sigma}| = 6$.
The word $\overline{\sigma}' = ababab\ldots$ is an element of Σ^ω: it is an infinite word over Σ and $|\overline{\sigma}'| = +\infty$. In the sequel, we will sometimes use ω-regular expressions to simply represent languages. Here, $\overline{\sigma}'$ can also be written $(ab)^\omega$.

Definition 2.7. *A* time sequence $\overline{\tau} = \tau_1\tau_2\tau_3\ldots$ *is a word over* $\mathbb{R}^+$ *such that* $\forall i < |\overline{\tau}|$, $\tau_i \leqslant \tau_{i+1}$.
A timed word *over the alphabet* Σ *is a pair* $\theta = (\overline{\sigma}, \overline{\tau})$ *where* $\overline{\sigma}$ *is a word over* Σ, $\overline{\tau}$ *a time sequence and* $|\overline{\sigma}| = |\overline{\tau}|$. *We also note* θ *as* $(\sigma_1, \tau_1)(\sigma_2, \tau_2)(\sigma_3, \tau_3)\ldots$, *and let* $|\theta| = |\overline{\sigma}|$.
We note ϵ *the empty timed word, we have* $|\epsilon| = 0$.
We denote by $T\Sigma^\star$ *the set of all finite timed words over* Σ *and by* $T\Sigma^\omega$ *the set of all infinite timed words over* Σ.

Definition 2.8. *A* language *is a (possibly infinite) set of words and a* timed language *is a (possibly infinite) set of timed words.*

Remark 2.9. In the sequel, we indifferently call *language* a set of finite words *or* a set of infinite words. In general, the kind of considered words will be clear from the context, otherwise, it will be specified.
The same remark holds for *timed languages.*

Example 2.10. Let us consider the alphabet $\Sigma = \{a, b\}$. Let us note $\overline{\sigma} = aaabab$ and $\overline{\tau} = 0.2\ 1\ 3.4\ 3.35\ 3.4\ 6$. $\theta = (\overline{\sigma}, \overline{\tau})$ is an element of $T\Sigma^\star$: it is a finite timed word over Σ. We can also denote θ as $(a, 0.2)(a, 1)(a, 3.4)(b, 3.35)(a, 3.4)(b, 6)$. The length of θ is equal to 6: $|\theta| = 6$.
Let us note $\overline{\sigma}' = ababab\ldots$ and $\overline{\tau}' = 0\ 1\ 2\ 3\ 4\ 5\ldots$. $\theta' = (\overline{\sigma}', \overline{\tau}')$ is an element of $T\Sigma^\omega$: it is an infinite timed word over Σ. θ' can also be denoted $\theta' = (a, 0)(b, 1)(a, 2)(b, 3)(a, 4)(b, 5)\ldots$ and $|\theta'| = +\infty$.
Let us note $\theta'' = (\overline{\sigma}'', \overline{\tau}'')$ with: $\forall i \geqslant 1$, $\sigma_i = a$; $\tau_1 = 0$ and $\forall j \geqslant 2$, $\tau_j = \tau_{j-1} + \frac{1}{2^j}$. θ'' is also a timed word.
Let us note $L_1 = \{\theta\}$: it is a timed language (of finite timed words) over Σ. The set L_2 of infinite timed words over Σ whose first letter is a is also a timed language (of infinite timed words): $L_2 = \{\theta = (\overline{\sigma}, \overline{\tau}) \mid \sigma_1 = a \ \wedge\ |\theta| = +\infty\}$. We have that: $\theta'' \in L_2$.
For the untimed setting, $L_3 = \{\overline{\sigma}'\}$ and $L_4 = \Sigma^\star$ are languages over Σ, respectively of infinite and finite words.

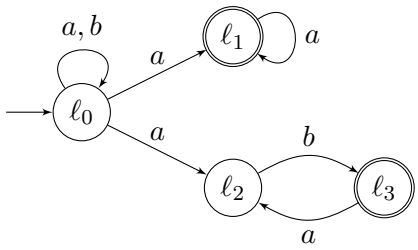


Figure 2.1: The automaton $\mathcal{B}$.

2.2 Automata

Before to consider the timed setting, we recall the simple notion of (untimed) automaton ([37]). It is a classical model dating from the fifties and having a lot of applications.

Here is the definition of the syntax of automata. We then distinguish automata accepting finite words and Büchi automata, that accept infinite words.

Definition 2.11. *An automaton (or a Büchi automaton (BA), when interpreted over infinite words) is a tuple* $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$*, where* Σ *is a finite* alphabet*,* L *is a finite set of* locations*,* $\ell_0 \in L$ *is the* initial location*,* $F \subseteq L$ *is the set of* accepting locations*, and* $\delta : L \times \Sigma \to 2^L$ *is the* transition function*.*

Example 2.12. As an example, consider the automaton $\mathcal{B}$ of Figure 2.1, over the alphabet $\Sigma = \{a, b\}$. It has four locations ℓ_0, ℓ_1, ℓ_2 and ℓ_3. We represent its initial location ℓ_0 with an ingoing arrow. The accepting locations from F are drawn with a double circle: $F = \{\ell_1, \ell_3\}$. The transition function of $\mathcal{B}$ is given by: $\delta(\ell_0, a) = \{\ell_0, \ell_1, \ell_2\}$, $\delta(\ell_0, b) = \{\ell_0\}$, $\delta(\ell_1, a) = \{\ell_1\}$, $\delta(\ell_1, b) = \varnothing$, $\delta(\ell_2, a) = \varnothing$, $\delta(\ell_2, b) = \{\ell_3\}$, $\delta(\ell_3, a) = \{\ell_2\}$ and $\delta(\ell_3, b) = \varnothing$.

As we will see thanks to the following definitions, an automaton is an acceptor of words.

Definition 2.13. *Let $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$ be an automaton. Let $\overline{\sigma} = \sigma_1\sigma_2\sigma_3\ldots$ be a (finite or infinite) word over Σ. We note $\ell \xrightarrow{\sigma} \ell'$ iff $\ell' \in \delta(\ell, \sigma)$. A* run *of $\mathcal{B}$ on $\overline{\sigma}$ is a (respectively, finite or infinite) sequence of transitions labelled by $\overline{\sigma}$, i.e. a sequence of the form: $\ell_0 \xrightarrow{\sigma_1} \ell_1 \xrightarrow{\sigma_2} \ell_2 \ldots \xrightarrow{\sigma_i} \ell_i \ldots$.*
A finite run is accepting *if its last configuration is accepting. An infinite run is* accepting *iff there are infinitely many $\ell_i \in F$ (i.e. we consider a Büchi acceptance condition).*
We say that a word $\overline{\sigma}$ is accepted *by $\mathcal{B}$ iff there is an accepting run of $\mathcal{B}$ on $\overline{\sigma}$.*

Definition 2.14. *We denote by $L(\mathcal{B})$ the set of finite words accepted by an automaton $\mathcal{B}$ and by $L^\omega(\mathcal{B})$ the set of infinite words accepted by $\mathcal{B}$. We call $L(\mathcal{B})$ and $L^\omega(\mathcal{B})$ the* languages *of $\mathcal{B}$, respectively over finite and infinite timed words.*

Example 2.15. Consider again the automaton $\mathcal{B}$ of Figure 2.1. Let us observe the finite word $\overline{\sigma} = baba$. Here are two possible runs of $\mathcal{B}$ on $\overline{\sigma}$:

$$\pi_1 = \ell_0 \xrightarrow{b} \ell_0 \xrightarrow{a} \ell_2 \xrightarrow{b} \ell_3 \xrightarrow{a} \ell_2$$
$$\pi_2 = \ell_0 \xrightarrow{b} \ell_0 \xrightarrow{a} \ell_0 \xrightarrow{b} \ell_0 \xrightarrow{a} \ell_1$$

π_1 is not accepting because $\ell_2 \notin F$. Nevertheless, π_2 is accepting (as $\ell_1 \in F$) and so $\overline{\sigma} \in L(\mathcal{B})$. It is easy to see that $L(\mathcal{B})$ consists in the finite words over Σ whose last letter is a or that finish by 'ab'. In a same way, $L^\omega(\mathcal{B})$ is the set of words $\sigma_1\sigma_2\sigma_3\ldots$ over Σ such that: there is $n_0 \geqslant 1$ such that for all $i \geqslant n_0$, $\sigma_i = a$; or there is $m_0 \geqslant 1$ such that for all $i \geqslant 0$, $\sigma_{m_0+(2i+1)} = a$ and $\sigma_{m_0+2i} = b$. Using ω-regular expressions:

$$L^\omega(\mathcal{B}) \;=\; (a+b)^\star a^\omega \;+\; (a+b)^\star (ab)^\omega.$$

An interesting characteristic of automata is the fact that they can be *deterministic*. A deterministic automaton has at most one run on each word.

Definition 2.16. *An automaton $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$ is deterministic iff for all $\ell \in L$, for all $\sigma \in \Sigma$, $|\delta(\ell, \sigma)| \leqslant 1$.*

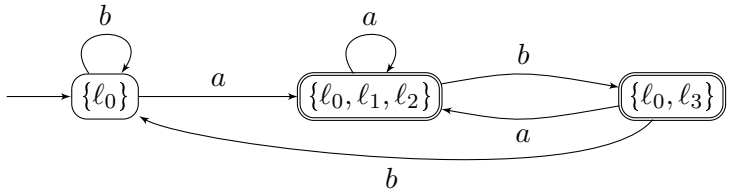


Figure 2.2: The automaton $\mathcal{B}_{det}$.

For each automaton $\mathcal{B}$, *interpreted over finite words*, we can construct a deterministic automaton $\mathcal{B}_{det}$ accepting the same language [55]. This can be done using a method called the *subset construction*. Indeed, the locations of the deterministic automaton constructed are *subsets* of locations of the original automaton. Here is the formal definition of $\mathcal{B}_{det}$:

Definition 2.17. *Let $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$ be an automaton. We define the deterministic automaton $\mathcal{B}_{det} = (\Sigma, L', \ell'_0, F', \delta')$, with:*

- $L' = 2^L$,
- $\ell'_0 = \{\ell_0\}$,
- $F' = \{S \subseteq L \mid S \cap F \neq \varnothing\}$,
- $\delta' : L' \times \Sigma \to L'$ *such that* $\delta'(S, \sigma) = \bigcup_{\ell^\star \in S} \delta(\ell^\star, \sigma)$.

Theorem 2.18. *For each automaton $\mathcal{B}$, the deterministic automaton $\mathcal{B}_{det}$ is such that: $L(\mathcal{B}_{det}) = L(\mathcal{B})$.*

Example 2.19. The automaton $\mathcal{B}_{det}$ of Figure 2.2 is the deterministic automaton obtained from the automaton $\mathcal{B}$ of Figure 2.1 using the subset construction.

However, this subset construction fails over infinite words, and, in general, Büchi automata are not *determinizable* ([37]):

Definition 2.20. *A Büchi automaton $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$ is said* determinizable *over infinite words if there is a deterministic Büchi automaton $\mathcal{B}_{det}$ such that $L^\omega(\mathcal{B}) = L^\omega(\mathcal{B}_{det})$.*

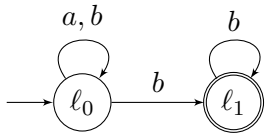


Figure 2.3: A non-determinizable Büchi automaton.

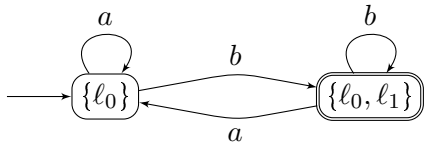


Figure 2.4: The automaton $\mathcal{D}$.

Theorem 2.21. *In general, Büchi automata are not determinizable.*

The Büchi automaton of Figure 2.3 recognizes the language of all infinite words containing a finite number of a's. There is no deterministic Büchi automaton recognizing this language.

Example 2.22. Let us consider the Büchi automaton of Figure 2.3. If we apply the subset construction on this automaton, we obtain the automaton $\mathcal{D}$ of Figure 2.4. $\mathcal{D}$ is deterministic but it does not recognize anymore the language of all infinite words containing a finite number of a's. For instance, $(ab)^\omega \in L^\omega(\mathcal{D})$.

This problem was overcome using another kind of automata called *Muller automata*. The structure of these automata is very close to that of Büchi automata, but they accept words in a different way.

Definition 2.23. *A Muller automaton is a tuple* $\mathcal{I} = (\Sigma, L, \ell_0, \mathcal{F}, \delta)$*, where* Σ *is a finite* alphabet*,* L *is a finite set of* locations*,* $\ell_0 \in L$ *is the* initial location*,* $\mathcal{F} \subseteq 2^L$ *is a familly of sets of accepting locations, and* $\delta : L \times \Sigma \to 2^L$ *is the* transition function.

A Muller automaton $\mathcal{I} = (\Sigma, L, \ell_0, \mathcal{F}, \delta)$ accepts an infinite word $\overline{\Sigma}$ if there is a run (see Definition 2.13) of $\mathcal{I}$ on $\overline{\sigma}$ such that the set of locations it goes

through infinitely often is precisely one $F \in \mathcal{F}$. The language $L^\omega(\mathcal{I})$ of a Muller automaton is the set of infinite words accepted by $\mathcal{I}$.

While a Büchi automaton is not determinizable in general, one can always find a deterministic Muller automaton recgnizing its language.

Theorem 2.24 ([59]). *For each Büchi automaton $\mathcal{B}$, there is a deterministic Muller automaton $\mathcal{I}$ such that $L^\omega(\mathcal{B}) = L^\omega(\mathcal{I})$.*

For a given automaton, a second interesting characteristic is the fact that we can construct an automaton representing the complement of its language, as defined below.

Definition 2.25. *Let $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$ be an automaton (respectively, be a Büchi automaton). We say that $\mathcal{B}$ is complementable if there is an automaton (respectively, a Büchi automaton) $\mathcal{B}^C$ such that $L(\mathcal{B}^C) = \Sigma^\star \backslash L(\mathcal{B})$ (respectively, such that $L(\mathcal{B}^C) = \Sigma^\omega \backslash L(\mathcal{B})$).*

Theorem 2.26 ([59]). *Each automaton over finite words and each Büchi automaton is complementable.*

Definition 2.27. *The emptiness problem asks, given an automaton (respectively, a Büchi automaton) $\mathcal{B}$, if $L(\mathcal{B}) = \varnothing$ (respectively, if $L^\omega(\mathcal{B}) = \varnothing$).*

Theorem 2.28 ([10]). *The emptiness problem is decidable for automata and Büchi automata.*

The emptiness problem for automata and Büchi automata can easily be solved using graphs traversals. Indeed, to verify if the finite word language of an automaton $\mathcal{C}$ is empty only consists in looking for a path to an accepting location of $\mathcal{C}$. Algorithm 1 answers the emptiness problem for automata over finite words. For a Büchi automaton $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$, another fix-point algorithm is used, to verify if a *reachable accepting location* of $\mathcal{B}$ is *reachable* from itself. Algorithm 2 answers the emptiness problem for automata over infinite words. It uses the following notations, for $\ell \subseteq L$, $S \subseteq L$ and $n \in \mathbb{N}$:

- $\delta(S) = \bigcup_{\ell \in S} \delta(\ell)$;
- $\delta^n(\ell) = \delta(\delta^{n-1}(\ell))$, with $\delta^0(\ell) = \ell$ and $\delta^1(\ell) = \delta(\ell)$;
- $\delta^\star(\ell) = \bigcup\limits_{n \in \mathbb{N}} \delta^n(\ell)$ and $\delta^+(\ell) = \bigcup\limits_{n \in \mathbb{N}_0} \delta^n(\ell)$.

Remark that, as L is finite, $\exists m \in \mathbb{N} : \delta^\star(\ell) = \bigcup_{n=1}^{m} \delta^n(\ell)$.

Algorithm 1 EmptinessAlgorithm

Input: An automaton $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$ over finite words.

Output: 'true' iff $L(\mathcal{B}) = \varnothing$.

```
S ← ℓ0
S_pre ← ∅
while S ≠ S_pre do
    if there is an accepting ℓ ∈ S then
        return 'false'
    end if
    S_pre ← S
    S ← S ∪ δ(S)
end while
return 'true'
```

Definition 2.29. *The universality problem asks, given an automaton (respectively, a Büchi automaton) $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$, if $L(\mathcal{B}) = \Sigma^\star$ (respectively, if $L^\omega(\mathcal{B}) = \Sigma^\omega$).*

Theorem 2.30. *The universality problem is decidable for automata and Büchi automata.*

This result is closely linked to the fact that automata and Büchi automata are complementable. Indeed, to decide the universality of a given automaton $\mathcal{B}$, we can compute the automaton $\mathcal{B}^C$, recognizing the complement of its language, and then use the algorithm solving the emptiness problem on $\mathcal{B}^C$.

Algorithm 2 EmptinessAlgorithmInfiniteWords

Input: A Büchi automaton $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$.

Output: 'true' iff $L(\mathcal{B}) = \varnothing$.

```
C ← ∅
D ← δ*(ℓ0) ∩ F
while C ≠ D do
    C ← D
    D ← δ+(D) ∩ F
end while
if D = ∅ then
    return true
else
    return false
end if
```

2.3 Alternating automata

Alternating automata ([37]) are an extension of automata. In classical automata, all transitions from a unique location, labelled by the same letter can be seen as *disjunctions*. For example, for the automaton of Figure 2.1, the transitions from location ℓ_0 with letter a could be described as $\delta(\ell_0, a) = \ell_0 \vee \ell_1 \vee \ell_2$. Alternating automata generalize plain automata in the sense that they admit *conjunctive transitions* as well as disjunctive ones: $\ell_0 \vee (\ell_1 \wedge \ell_2)$ could also be a transition, for an alternating automaton. Intuitively, when taking a conjunctive transition, an alternating automaton creates several copies of itself that run in parallel and must *all* accept the suffix of the word.

Syntax of alternating automata. To give the syntax of alternating automata, we start by defining *positive boolean formulas*, which are used in the transition function of alternating automata.

Definition 2.31. *Let L be a finite set. The set of positive boolean formulas over*

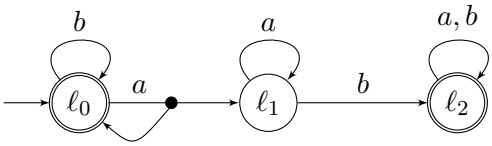


Figure 2.5: Alternating automaton $\mathcal{A}$

L, denoted by $B^+(L)$, contains $\top$, $\bot$, all elements from L, and all finite boolean combinations over L built with $\wedge$ and $\vee$.

Definition 2.32. *An **alternating automaton** is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ where Σ is a finite alphabet, L is a finite set of locations, ℓ_0 is the initial location, $F \subseteq L$ is a set of accepting locations, $\delta : L \times \Sigma \to B^+(L)$ is the transition function.*

Example 2.33. As an example, consider the alternating automaton $\mathcal{A}$ in Figure 2.5, over the alphabet $\Sigma = \{a, b\}$. $\mathcal{A}$ has three *locations* ℓ_0, ℓ_1 and ℓ_2, such that ℓ_0 is initial and ℓ_0 and ℓ_2 are final. Its transition function is given by: $\delta(\ell_0, a) = \ell_0 \wedge \ell_1$, $\delta(\ell_0, b) = \ell_0$, $\delta(\ell_1, a) = \ell_1$, $\delta(\ell_1, b) = \ell_2$ and $\delta(\ell_2, a) = \delta(\ell_2, b) = \ell_2$. Observe that, in the figure, we represent the (conjunctive) transition $\delta(\ell_0, a) = \ell_0 \wedge \ell_1$ by an arrow splitting in two branches connected respectively to ℓ_0 and ℓ_1. Intuitively, *taking this transition* means that, when reading an a from location ℓ_0, the automaton should start *two copies of itself*, one in location ℓ_0, and a second in location ℓ_1. Both copies should accept the suffix for the word to be accepted (this notion will be defined formally in a following paragraph).

Semantics of alternating automata. We now explain the semantics of alternating automata. We start by defining what is a *configuration* of an alternating automaton and how it can move from one configuration to another using the notion of *minimal model*.

Definition 2.34. *A* configuration *of an alternating automaton $\mathcal{A}$ is a (possibly empty) finite set of locations of $\mathcal{A}$. The* initial configuration *of $\mathcal{A}$ is $\{\ell_0\}$. A*

configuration is accepting *iff all the locations it contains are accepting.*
We note $\mathrm{Config}(\mathcal{A})$ *the set of all configurations of* $\mathcal{A}$.

Definition 2.35. *Let L be a finite set. A subset M of L is a* model *of $\gamma \in B^+(L)$ if the truth assignment that assigns $\top$ to the elements of M and $\bot$ to the elements of $L \backslash M$ satisfies γ. We say that M is a* minimal *model of γ iff M is a model of γ and no proper subset of M is a model of γ.*

Intuitively, for $\ell \in L$ and $\sigma \in \Sigma$, a minimal model of $\delta(\ell, \sigma)$ represents one configuration that the automaton can reach from location ℓ by reading σ.

Example 2.36. Let us consider again the alternating automaton of Figure 2.5. A model M of $\delta(\ell_0, a) = \ell_0 \wedge \ell_1$ must at least contain locations ℓ_0 and ℓ_1. Indeed, if M does not contain ℓ_0 or ℓ_1, the truth assignment that assigns $\top$ to the elements of M and $\bot$ to the elements of $L \backslash M$ does not satisfy $\ell_0 \wedge \ell_1$. So, the models of $\delta(\ell_0, a)$ are $\{\ell_0, \ell_1\}$ and $\{\ell_0, \ell_1, \ell_2\}$; and its unique minimal model is $\{\ell_0, \ell_1\}$.

We now define some notations enabling to simply express what are the possible successors of a given configuration of an alternating automaton.

Definition 2.37. *We denote* $\mathsf{Succ}(\ell, \sigma) = \{M \mid M$ *is a minimal model of* $\delta(\ell, \sigma)\}$. *We lift the definition of* Succ *to configurations C as follows:* $\mathsf{Succ}(C, \sigma)$ *is the set of all configurations C' of the form $\cup_{\ell \in C} M_\ell$, where, for all $\ell \in C$: $M_\ell \in \mathsf{Succ}(\ell, \sigma)$. That is, each $C' \in \mathsf{Succ}(C, \sigma)$ is obtained by choosing one minimal model M_ℓ in* $\mathsf{Succ}(\ell, \sigma)$ *for each $\ell \in C$, and taking the union of all those M_ℓ.*

Example 2.38. Let us consider the alternating automaton of Figure 2.6 and its configuration $C = \{\ell_0, \ell_4\}$. We want to compute $\mathsf{Succ}(C, a)$. We start by calculating $\mathsf{Succ}(\ell_0, a)$ and $\mathsf{Succ}(\ell_4, a)$. The minimal models of $\delta(\ell_0, a) = (\ell_0 \wedge \ell_1) \vee (\ell_0 \wedge \ell_4) \vee \ell_3$ are $C_1 = \{\ell_0, \ell_1\}$, $C_2 = \{\ell_0, \ell_4\}$ and $C_3 = \{\ell_3\}$. So, $\mathsf{Succ}(\ell_0, a) = \{C_1, C_2, C_3\}$. The unique minimal model of $\delta(\ell_4, a) = \ell_1$ is $D_1 = \{\ell_1\}$. So, $\mathsf{Succ}(\ell_0, a) = \{D_1\}$. Hence, $\mathsf{Succ}(C, a) = \{C_1 \cup D_1, C_2 \cup D_1, C_3 \cup D_1\} = \{\{\ell_0, \ell_1\}, \{\ell_0, \ell_4, \ell_1\}, \{\ell_3, \ell_1\}\}$.

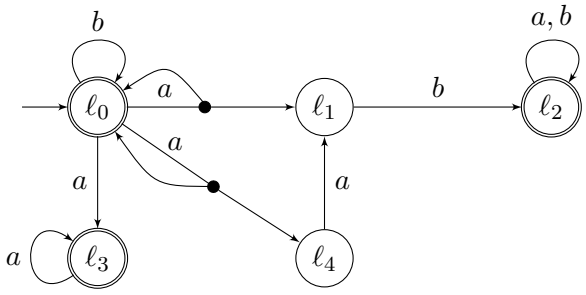


Figure 2.6: Alternating automaton $\mathcal{A}$

Runs of alternating automata. We can now formally define the notion of *run* of an alternating automaton. Each new configuration in the run is obtained by performing a *discrete step*, which characterizes the *semantics* of an alternating automaton. The definition of run is based on this discrete step.

Definition 2.39. *Let $\mathcal{A}$ be an alternating automaton. The* semantics *of $\mathcal{A}$ is the transition system $\mathcal{T}_{\mathcal{A}} = (\mathrm{Config}\,(\mathcal{A})\,, \longrightarrow)$ on the configurations of $\mathcal{A}$ defined as follows. The transition relation $\longrightarrow$ takes care of discrete transitions between locations: $C \xrightarrow{\sigma} C'$ iff $C' \in \mathsf{Succ}(C, \sigma)$. We let $\longrightarrow = \bigcup_{\sigma \in \Sigma} \xrightarrow{\sigma}$.*

Definition 2.40. *Let $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ be an alternating automaton and let $\overline{\sigma}$ be a finite or infinite word. A* run *of $\mathcal{A}$ on $\overline{\sigma}$ is a finite or infinite sequence of discrete transitions in $\mathcal{T}_{\mathcal{A}}$ that is labelled by $\overline{\sigma}$, i.e. a sequence of the form: $C_0 \xrightarrow{\sigma_1} C_1 \xrightarrow{\sigma_2} C_2 \ldots \xrightarrow{\sigma_n} C_n \ldots$.*

Observe that for all pairs of configurations C, C' such that $C' \in \mathsf{Succ}(C, \sigma)$ for some σ, each $\ell \in C$ can be associated with a unique set $\mathsf{dest}(C, C', \ell) \subseteq C'$ containing all the 'successors' of ℓ in C' and obtained as follows. By definition, $C' = \bigcup_{\overline{\ell} \in C} M_{\overline{\ell}}$, where each $M_{\overline{\ell}} \in \mathsf{Succ}(\overline{\ell}, \sigma)$ is the minimal model that has been chosen for $\overline{\ell}$ when computing $\mathsf{Succ}(C, \sigma)$. Then, we let $\mathsf{dest}(C, C', \ell) = M_{\ell}$.

The function dest allows to define a DAG representation of runs. We regard a run π as a rooted DAG $G_{\pi} = (V, \rightarrow)$, whose vertices V correspond to the

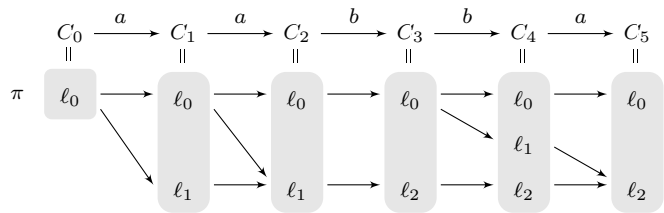


Figure 2.7: Representations of a run of an alternating automaton.

locations of the alternating automaton (vertices at depth i correspond to C_i), and whose set of edges $\rightarrow$ expresses the transitions of the alternating automaton. Here is its formal definition.

Definition 2.41. *Let $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ be an alternating automaton and let $\pi = C_0 \xrightarrow{\sigma_1} C_1 \xrightarrow{\sigma_2} C_2 \ldots \xrightarrow{\sigma_n} C_n \ldots$ be a run of $\mathcal{A}$ on $\overline{\sigma}$. We define the rooted DAG $G_\pi = (V, \rightarrow)$ with:*

- *$V = \bigcup_{0 \leqslant i \leqslant |\overline{\sigma}|} V_i$, where for all $0 \leqslant i \leqslant |\overline{\sigma}|$: $V_i = \{(\ell, i) \mid \ell \in C_i\}$ is the set of all vertices of depth i ;*
- *the root of G_π is $(\ell_0, 0)$; and*
- *$(\ell_1, i_1) \rightarrow (\ell_2, i_2)$ iff $i_2 = i_1 + 1$ and $\ell_2 \in \mathsf{dest}(C_{i_1}, C_{i_2}, \ell_1)$.*

Example 2.42. Figure 2.7 displays the two possible representations of a run π of the alternating automaton $\mathcal{A}$ in Figure 2.5, on the word $a\,a\,b\,a\,b$: grey boxes highlight the successive configurations. The edges of the DAG of the run Definition 2.41 are obtained observing function dest. For instance, the edges linking locations of C_1 to locations of C_2 are obtained thanks to $\mathsf{dest}(C_1, C_2, \ell_0)$ and $\mathsf{dest}(C_1, C_2, \ell_1)$. We have: $C_2 = \bigcup_{\ell \in C_1} M_\ell$, for the minimal models $M_{\ell_0} = \{\ell_0, \ell_1\}$ of $\delta(\ell_0, a)$, and $M_{\ell_1} = \{\ell_1\}$ of $\delta(\ell_1, a)$. Hence, $\mathsf{dest}(C_1, C_2, \ell_0) = M_{\ell_0} = \{\ell_0, \ell_1\}$: there is an edge from ℓ_0 in C_1 to locations ℓ_0 and ℓ_1 in C_2. In a similar way, $\mathsf{dest}(C_1, C_2, \ell_0) = M_{\ell_1} = \{\ell_1\}$: there is an edge from ℓ_1 in C_1 to location ℓ_1 in C_2.

Language of alternating automata. We can now define the accepted language of an alternating automaton, over finite and infinite words.

For finite words, the characterisation of runs given by Definition 2.40 is more convenient:

Definition 2.43. *A finite run is* accepting *iff its last configuration C_n is accepting and we say that a finite word is accepted by $\mathcal{A}$ iff there exists an accepting finite run of $\mathcal{A}$ on this word. We note $L(\mathcal{A})$ the language of all finite words accepted by $\mathcal{A}$.*

Nevertheless, to define when an infinite word is accepted by $\mathcal{A}$, we need to use the DAG characterisation of runs.

Definition 2.44. *We call* branch *of a run represented by a DAG G_π a (finite or) infinite path in G_π. We note $Bran^\omega(G_\pi)$ the set of all* infinite *branches of G_π and, for $\beta \in Bran^\omega(G_\pi)$, we note $Inf(\beta)$ the set of locations occurring infinitely often along β.*
The run represented by G_π is accepting *iff $\forall \beta \in Bran^\omega(G_\pi)$, $Inf(\beta) \cap F \neq \emptyset$ (i.e. we consider a Büchi acceptance condition). We say that an infinite word θ is accepted by $\mathcal{A}$ iff there exists an accepting run of $\mathcal{A}$ on θ.*
We note $L^\omega(\mathcal{A})$ the language of all infinite words accepted by $\mathcal{A}$.

Example 2.45. Let us observe again the run π given in Figure 2.7, for the alternating automaton $\mathcal{A}$ of Figure 2.5, on the finite word $\overline{\sigma} = a\,a\,b\,b\,a$. It is accepting because C_5 is accepting. If $\overline{\sigma}$ is extended by an infinite number of b's, we obtain an infinite word also accepted by $\mathcal{A}$. Indeed, the run of $\mathcal{A}$ on this infinite word has π as prefix and the two unique branches of this run will infinitely keep looping on ℓ_0 and ℓ_2, which are accepting locations. $L(\mathcal{A})$ is the set of finite words on $\Sigma = \{a, b\}$ where each a is eventually followed by a b. On the other hand, $L^\omega(\mathcal{A})$ is the set of infinite words on Σ satisfying the same property.

From alternating automata to automata. Classical methods enable to translate an alternating automaton into an automaton recognizing the same language.

Over finite words, this translation from an alternating automaton $\mathcal{A}$ to an automaton $\mathcal{B}^{\mathcal{A}}$ uses a *subset construction* very close to that enabling to determinize an automaton over finite words. The main difference is that a powerset location is now accepting if the locations of the alternating automaton it contains are *all* accepting. Here is the formal definition of $\mathcal{B}^{\mathcal{A}}$:

Definition 2.46. *Let $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ be an alternating automaton. We define the automaton $\mathcal{B}^{\mathcal{A}} = (\Sigma, L', \ell'_0, F', \delta')$, with:*

- $L' = 2^L$,
- $\ell'_0 = \{\ell_0\}$,
- $F' = 2^F$,
- $\delta' : L' \times \Sigma \to 2^{L'}$ *such that* $S' \in \delta'(S, \sigma)$ *iff* $S' = \bigcup_{\ell \in S} M^\ell$, *with,* $\forall \ell \in S$, *a minimal model* M^ℓ *of* $\delta(\ell, \sigma)$.

Example 2.47. Let us consider the alternating automaton $\mathcal{A}$ of Figure 2.8. $L(\mathcal{A})$ is the set of finite words on $\Sigma = \{a, b, c\}$ satisfying the following property: the first letter of the word is an a, each c it contains is eventually followed by a b and each b it contains is eventually followed by an a. The subset construction of definition 2.46, applied to $\mathcal{A}$ gives the automaton $\mathcal{B}^{\mathcal{A}}$ (over finite words) of Figure 2.9 with $L(\mathcal{A}) = L(\mathcal{B}^{\mathcal{A}})$.

Nevertheless, when considering a Büchi alternating automaton (over infinite words), a more involved method is required to turn it into a non-deterministic Büchi automaton. Indeed, the acceptance condition observes the branches of the DAG representation of a run: all of them must infinitely often visit F. The problem with this acceptance condition is that we cannot decide if a run of $\mathcal{A}$ is accepting only observing the sequence of its configurations. Instead, we must take into account the different branches to be sure all of them visit F infinitely often. Miyano and Hayashi have proposed such a translation ([46]) using markers (a marker for each location forming a configuration) to distinguish branches on

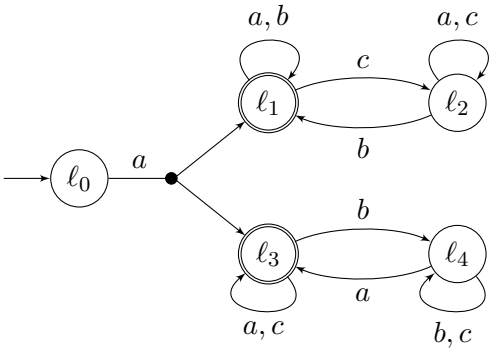


Figure 2.8: Alternating automaton $\mathcal{A}$

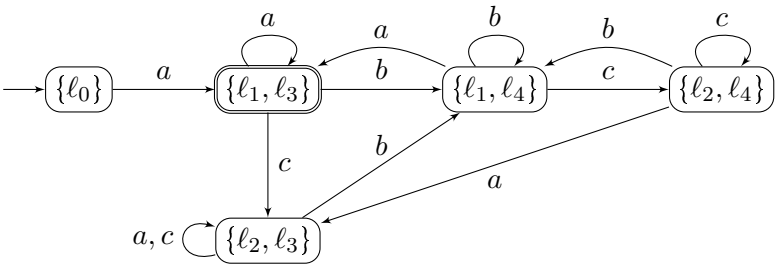


Figure 2.9: Automaton $\mathcal{B}^{\mathcal{A}}$ such that $L(\mathcal{A}) = L(\mathcal{B}^{\mathcal{A}})$

which F has already been visited from others. Each location of a configuration of $\mathcal{A}$ is marked by $\top$ or $\bot$ if, respectively, the run has visited F on the branches leading to this location or not. When all the branches have visited F (i.e. all the locations of the configuration reached are marked by $\top$), we reach an *accepting configuration*: we then reset all the markings to $\bot$. Hence, if we encounter infinitely many such accepting configurations we are sure that each branch visit F infinitely often.

Example 2.48. Let us consider again the alternating automaton $\mathcal{A}$ of Figure 2.8. $L^\omega(\mathcal{A})$ is the set of infinite words on $\Sigma = \{a, b, c\}$ such that the first letter of the word is an a, each c it contains is eventually followed by a b and each b it contains is eventually followed by an a. The Miyano-Hayashi construction gives the Büchi automaton $\mathcal{B}^{MH}$ of Figure 2.10: $L^\omega(\mathcal{A}) = L^\omega(\mathcal{B}^{MH})$.

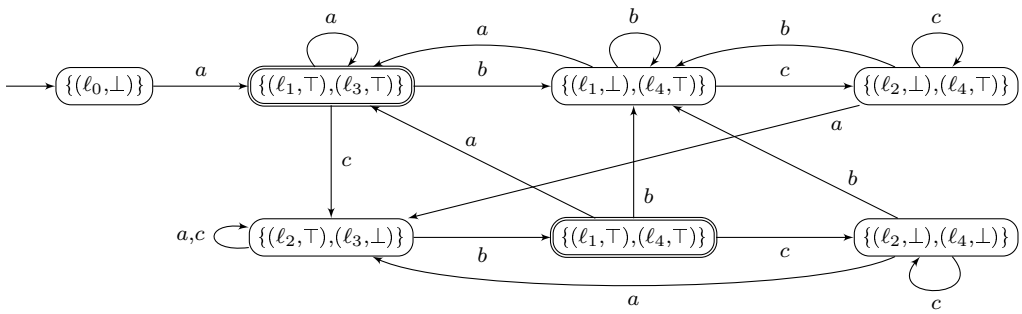


Figure 2.10: Büchi automaton $\mathcal{B}^{MH}$ such that $L^\omega(\mathcal{A}) = L^\omega(\mathcal{B}^{MH})$

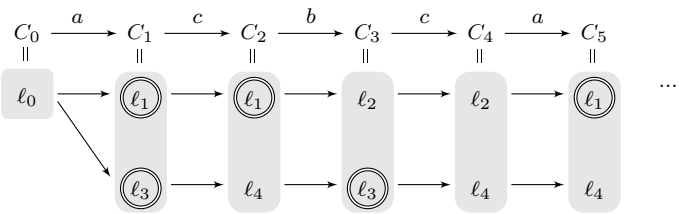


Figure 2.11: Run on the word *acbca*.

Remark that the infinite word $\overline{\sigma} = ac(abc)^\omega$ is in $L^\omega(\mathcal{A})$. This word is indeed accepted by $\mathcal{B}^{MH}$. Observe that, the automaton $\mathcal{B}^{\mathcal{A}}$ (of Figure 2.9), given by the subset construction, does not accept $\overline{\sigma}$ when interpreted as a Büchi automaton. It is due to the fact that $\mathcal{A}$ accepts a finite word if it reaches ℓ_1 and ℓ_3 at the same time. Instead, for an infinite word to be accepted, locations ℓ_1 and ℓ_3 may be reached in a desynchronized manner, as is the run of $\mathcal{A}$ on $\overline{\sigma}$ (see Figure 2.11). The automaton $\mathcal{B}^{MH}$ obtained by the Miyano-Hayashi method takes this fact into account.

2.4 Linear Temporal Logic

Temporal logics are used to represented subsets of words over a given alphabet, as well as automata. In particular, Linear Temporal Logic (LTL) is convenient to specify desired *properties* over words.

2.4.1 The LTL syntax and semantics

First, let us formally define the syntax of LTL.

Definition 2.49 (LTL syntax)**.** *Given a finite alphabet Σ, the formulas of LTL are defined by the following syntax, where $\sigma \in \Sigma$:*

$$\Phi := \top \quad | \quad \sigma \quad | \quad \Phi_1 \wedge \Phi_2 \quad | \quad \neg\Phi \quad | \quad \Phi_1 U_I \Phi_2.$$

In the sequel, we rely on the following usual shortcuts. The 'eventually' operator: $\Diamond\Phi$ stands for $\top U \Phi$. The 'always' operator: $\Box\Phi$ stands for $\neg\Diamond\neg\Phi$. We will also use $\Phi_1 \tilde{U} \Phi_2$ for $\neg(\neg\Phi_1 U \neg\Phi_2)$.

We now define the LTL semantics. We will see that this semantics is adapted for the setting of finite words as well as for that of infinite words.

Definition 2.50 (LTL semantics)**.** *Given a word $\overline{\sigma}$ over Σ, a position $1 \leqslant i \leqslant |\overline{\sigma}|$ and an LTL formula Φ, we say that $\overline{\sigma}$ satisfies Φ from position i, written $(\overline{\sigma}, i) \models \Phi$ iff the following holds:*

- $(\overline{\sigma}, i) \models \top$,
- $(\overline{\sigma}, i) \models \sigma$ *iff* $\sigma_i = \sigma$,
- $(\overline{\sigma}, i) \models \Phi_1 \wedge \Phi_2$ *iff* $(\overline{\sigma}, i) \models \Phi_1$ *and* $(\overline{\sigma}, i) \models \Phi_2$,
- $(\overline{\sigma}, i) \models \neg\Phi$ *iff* $(\overline{\sigma}, i) \not\models \Phi$,

- $(\overline{\sigma}, i) \models \Phi_1 U \Phi_2$ *iff*
  $\exists i \leqslant j \leqslant |\theta|$, *such that* $(\overline{\sigma}, j) \models \Phi_2$ *and* $\forall i \leqslant k < j, (\overline{\sigma}, k) \models \Phi_1$.

We say that $\overline{\sigma}$ satisfies Φ, *written* $\overline{\sigma} \models \Phi$, *iff* $(\overline{\sigma}, 1) \models \Phi$. *We note*

$$[\![\Phi]\!] := \{\overline{\sigma} \mid |\overline{\sigma}| < \infty \text{ and } \overline{\sigma} \models \Phi\}$$

the finite word language *of* Φ *and*

$$[\![\Phi]\!]^{\omega} := \{\overline{\sigma} \mid |\overline{\sigma}| = +\infty \text{ and } \overline{\sigma} \models \Phi\}$$

the infinite word language *of* Φ.

Example 2.51. LTL enables to express *qualitative* properties, on the sequence of events. For instance, the LTL formula $\Phi \equiv \Box(p \Rightarrow \Diamond q)$ expresses that 'each p will be eventually followed by a q'.

We still need to define what is a *subformula* of a given LTL formula.

Definition 2.52. *Let* Φ *be an LTL formula, we note* $Sub(\Phi)$ *the set of all subformulas of* Φ, *i.e.:*

- $Sub\,(\Phi) = \{\Phi\}$ *when* $\Phi \in \{\top\} \cup \Sigma$,
- $Sub\,(\neg\Phi) = \{\neg\Phi\} \cup Sub\,(\Phi)$ *and*
- $Sub\,(\Phi) = \{\Phi\} \cup Sub\,(\Phi_1) \cup Sub\,(\Phi_2)$ *when either* $\Phi \equiv \Phi_1 U_I \Phi_2$ *or* $\Phi \equiv \Phi_1 \wedge \Phi_2$.

We define the *size* of an LTL formula as follows:

Definition 2.53. *Let* Φ *be an LTL formula. We let* $|\Phi|$ *denote the* size of Φ, *defined as the* number *of* U *modalities it contains.*

2.4.2 The problems

In this section, we define the problems on LTL we are interested in. We also present the decidability and complexity results concerning them.

Satisfiability. The first problem on the LTL logic consists in verifying that there exists a word satisfying a given LTL formula.

Definition 2.54. *Let Φ be an LTL formula. We say that Φ is satisfiable over finite (respectively infinite) words iff there exists a finite (respectively infinite) word $\overline{\sigma}$ such that $\overline{\sigma}$ satisfies Φ.*

Definition 2.55. *The satisfiability problem over finite (respectively infinite) words is the problem of deciding if a given LTL formula Φ is satisfiable over finite (respectively infinite) words.*

Algorithmically, to solve the LTL satisfiability problem consists in verifying if $[\![\Phi]\!] \neq \emptyset$ (or $[\![\Phi]\!]^\omega \neq \emptyset$, over infinite words), see Definition 2.27. In general, over finite words, an automaton $\mathcal{B}_\Phi$ recognizing $[\![\Phi]\!]$ is constructed and the algorithm consists in looking for a path to an accepting location of $\mathcal{B}_\Phi$. Over infinite word, a fix-point algorithm is used to verify if a *reachable accepting* location of $\mathcal{B}_\Phi$ is *reachable from itself* (see Algorithms 1 and 2).

Definition 2.56 ([25])**.** *The LTL satisfiability problems, over finite and infinite words, are PSPACE-complete.*

Model-Checking. The main problem on the LTL logics we are interested in is the 'LTL Mordel-Checking problem' ([61, 63]). Being given an automaton or a Büchi automaton $\mathcal{B}$ and an LTL formula Φ, it consists in verifying if the language of $\mathcal{B}$ is included in the language of Φ. In other words, it enables to verify if each accepting run of $\mathcal{B}$ satisfies the property described by Φ.

Definition 2.57. *Let $\mathcal{C}$ be an automaton and Φ be an LTL formula. The LTL model-checking problem over finite words is to decide if the following inclusion*

holds: $L(\mathcal{C}) \subseteq [\![\Phi]\!]$.
Symmetrically, let $\mathcal{B}$ *be a Büchi automaton and* Φ *be an LTL formula. The LTL model-checking problem over infinite words is to decide if the following inclusion holds:* $L^\omega(\mathcal{B}) \subseteq [\![\Phi]\!]^\omega$.

Definition 2.58 ([61]). *The LTL model-checking problem is PSPACE-complete.*

Classical algorithms to solve the LTL model-checking problem work as follows. We first remark that the inclusion $L(\mathcal{B}) \subseteq [\![\Phi]\!]$ is indeed equivalent to verify that $L(\mathcal{B}) \cap [\![\neg\Phi]\!] = \varnothing$; as well as $L^\omega(\mathcal{B}) \subseteq [\![\Phi]\!]^\omega$ is equivalent to $L^\omega(\mathcal{B}) \cap [\![\neg\Phi]\!]^\omega = \varnothing$. So, these algorithms consist in negating Φ and then constructing an automaton (respectively, a Büchi automaton) $\mathcal{B}_{\neg\Phi}$ that recognizes $[\![\neg\Phi]\!]$ (respectively, $[\![\neg\Phi]\!]^\omega$) in way to verify that $L(\mathcal{B}) \cap L(\mathcal{B}_{\neg\Phi}) = \varnothing$ (respectively, $L^\omega(\mathcal{B}) \cap L^\omega(\mathcal{B}_{\neg\Phi}) = \varnothing$). The last step is to construct an automaton (respectively, a Büchi automaton) $\mathcal{B} \times \mathcal{B}_{\neg\Phi}$ such that $L(\mathcal{B} \times \mathcal{B}_{\neg\Phi}) = L(\mathcal{B}) \cap L(\mathcal{A}_{\neg\Phi})$ (respectively, $L^\omega(\mathcal{B} \times \mathcal{B}_{\neg\Phi}) = L^\omega(\mathcal{B}) \cap L^\omega(\mathcal{A}_{\neg\Phi})$) and verify if its language is empty. Those model-checking methods have reached a high maturity level, as can be seen from the various industrial and academical existing tools, such as SPIN and nuSMV ([24, 58]).
Remark that, these algorithms finally solve the LTL model-checking problem verifying if $L(\mathcal{B} \times \mathcal{B}_{\neg\Phi}) = \varnothing$ (or $L^\omega(\mathcal{B} \times \mathcal{B}_{\neg\Phi}) = \varnothing$). This is why the algorithms to solve the LTL model-checking and LTL satisfiability problems are very close to each other: once they are given $\mathcal{B} \times \mathcal{B}_{\neg\Phi}$ or $\mathcal{B}_\Phi$, they uses the same techniques to verify if its language is empty or not.

More recently, a new technique, that proved its efficiency in practice, was proposed ([34]) to solve the LTL model-checking problem *on the fly*. To verify if $L(\mathcal{B}) \cap [\![\neg\Phi]\!] = \varnothing$, an *alternating automaton* $\mathcal{A}_{\neg\Phi}$ recognizing $[\![\neg\Phi]\!]$ is first constructed. The interest is that such an alternating automaton has a size linear in the size of $\neg\Phi$, while an automaton recognizing $[\![\neg\Phi]\!]$ has in general a size exponential in the size of $\neg\Phi$. The aim is then to verify if $L(\mathcal{B}) \cap L(\mathcal{A}_{\neg\Phi}) = \varnothing$. This is done constructing *on the fly* an automaton $\mathcal{B} \times \mathcal{A}_{\neg\Phi}$ such that $L(\mathcal{B} \times$

$\mathcal{A}_{\neg\Phi}) = L(\mathcal{B}) \cap L(\mathcal{A}_{\neg\Phi})$. *On the fly* means the locations and transitions of $\mathcal{B} \times \mathcal{A}_{\neg\Phi}$ are constructed one by one, from the initial location of $\mathcal{B} \times \mathcal{A}_{\neg\Phi}$. The construction can be stopped as soon as a path leading to an accepting location is found: it proves that $L(\mathcal{B}) \cap L(\mathcal{A}_{\neg\Phi}) \neq \emptyset$. In practice, this *on the fly* algorithm is more efficient than the classical one because the construction of the whole automaton $\mathcal{B} \times \mathcal{A}_{\neg\Phi}$ might be avoided.
A similar method also exists in the setting of infinite words ([34]): a fix-point algorithm is used to verify *on the fly* if the part of $\mathcal{B} \times \mathcal{A}_{\neg\Phi}$ yet constructed contains an accepting run, i.e. if a *reachable* accepting location of the constructed part of $\mathcal{B} \times \mathcal{A}_{\neg\Phi}$ is *reachable from itself.* In such a case, the construction can be stopped and the LTL model-checking algorithm answers $L^\omega(\mathcal{B}) \nsubseteq [\![\Phi]\!]^\omega$.

In the following subsection, we will formally explain the translation from an LTL formula to an alternating automaton. This is the basis construction of this efficient algorithm solving the LTL model-checking problem. We will use a similar construction to solve the model-checking problem of another logic, namely *MITL*, investigated in our contributions.

Reactive synthesis. We have just seen that model-checking algorithms enable to verify that all the executions of a given system, represented by a (Büchi) timed automaton, satisfy desired properties, expressed by LTL formulas. Nevertheless, when the system does not satisfy the given property, it is erroneous and must be designed again. The reactive synthesis aims to avoid this trial-and-error approach.

The LTL reactive synthesis problem [1, 32, 54] considers a property, given by an LTL formula. It is formalized by a game played between two players: the *environment* and the *controller*. The aim of the controller is to manage to build a system satisfying the given property, despite the constraints of the reality, represented by the *environment*. To solve the LTL reactive synthesis problem consists in computing a controller that, when composed with the environment, will satisfy the property.

The LTL reactive synthesis is played on a particular semanics of LTL in which letters are replaced by *atomic propositions*.

Definition 2.59. *The LTL reactive synthesis is formalized by the following game. Given an LTL formula Φ and a partition of its atomic propositions AP into C and E, the controller starts by giving a subset $C_0 \subseteq C$ of propositions, the environment responds by giving a subset of propositions $E_0 \subseteq E$, then the controller gives $C_1 \subseteq C$ and the environment responds by $E_1 \subseteq E$, and so on. This game lasts forever and the outcome of the game is the infinite word $\bar{\sigma} = (C_0 \cup E_0)(C_1 \cup E_1)(C_2 \cup E_2) \cdots \in (2^{AP})^\omega$. The controller wins if the resulting infinite word $\bar{\sigma} \in [\![\Phi]\!]^\omega$. The LTL reactive synthesis problem asks to produce a winning strategy for the controller.*

The complexity of the LTL reactive synthesis problem was studied in [54].

Theorem 2.60 ([54])**.** *The LTL reactive synthesis is 2EXPTIME-complete.*

2.5 From LTL to Alternating Automata

We recall the translation from an LTL formula Φ to an alternating automaton $\mathcal{A}_\Phi$ recognizing $[\![\Phi]\!]$. This translation is the basis of efficient algorithms solving the LTL model-checking problem. We present here the translation due to Ouaknine and Worrell ([51]). This translation was proposed for the Metric Temporal Logic (MTL) in [51] and includes *timed aspects*. We chose to present it for the sake of continuity because we will use it on the entire MTL in a following section. For the present setting, we removed the timed aspects of the definition of [51] in way to make it correspond to plain LTL formulas. There exists other translations, slightly differing from the present one, also used in such algorithms ([34, 45, 62]).

To formally define $\mathcal{A}_\Phi$, observe that we can transform any LTL formula in an equivalent LTL formula in *negative normal form* (in which negation can only be

present on letters $\sigma \in \Sigma$), of the same size, using the operators: $\wedge, \vee, \neg, U$ and $\tilde{U}$.

Example 2.61. The LTL formula $\Phi \equiv \Box(a \Rightarrow \Diamond b)$ is equivalent to the formula $\bot\tilde{U}(\neg a \vee \top U b)$, which is in negative normal form. Its negation, $\neg\big(\Box(a \Rightarrow \Diamond b)\big)$, is equivalent to the following negative normal form formula: $\top U(a \wedge \bot\tilde{U}\neg b)$.

Definition 2.62. *For an LTL formula Φ in negative normal form, we let $\mathcal{A}_\Phi = (\Sigma, L, \ell_0, F, \delta)$ where: L is the set containing the initial copy of Φ, noted 'Φ_{init}', and all the formulas of $Sub(\Phi)$ whose outermost connective is 'U' or '$\tilde{U}$'; $\ell_0 = \Phi_{init}$; F is the set of the elements of L of the form $\Phi_1\tilde{U}_I\Phi_2$. Finally δ is defined by induction on the structure of Φ:*

- $\delta(\Phi_{init}, \sigma) = \delta(\Phi, \sigma)$
- $\delta(\Phi_1 \vee \Phi_2, \sigma) = \delta(\Phi_1, \sigma) \vee \delta(\Phi_2, \sigma)$
- $\delta(\Phi_1 \wedge \Phi_2, \sigma) = \delta(\Phi_1, \sigma) \wedge \delta(\Phi_2, \sigma)$
- $\delta(\Phi_1 U \Phi_2, \sigma) = \delta(\Phi_2, \sigma) \vee (\delta(\Phi_1, \sigma) \wedge \Phi_1 U \Phi_2)$
- $\delta(\Phi_1\tilde{U}\Phi_2, \sigma) = \delta(\Phi_2, \sigma) \wedge (\delta(\Phi_1, \sigma) \vee \Phi_1\tilde{U}\Phi_2)$
- $\forall \sigma_1, \sigma_2 \in \Sigma$:
  $\delta(\sigma_1, \sigma_2) = \begin{cases} \textit{true} & \textit{if } \sigma_1 = \sigma_2 \\ \textit{false} & \textit{if } \sigma_1 \neq \sigma_2 \end{cases}$ *and* $\delta(\neg\sigma_1, \sigma_2) = \begin{cases} \textit{false} & \textit{if } \sigma_1 = \sigma_2 \\ \textit{true} & \textit{if } \sigma_1 \neq \sigma_2 \end{cases}$
- $\forall \sigma \in \Sigma$: $\delta(\top, \sigma) = \top$ *and* $\delta(\bot, \sigma) = \bot$.

Example 2.63. Let us consider the LTL formula $\Phi \equiv \Box(a \Rightarrow \Diamond b)$, which is a shorthand for $\bot\tilde{U}\big(a \Rightarrow (\top U b)\big)$. The alternating automaton $\mathcal{A}_\Phi$ is given in Figure 2.12, where the location $\ell_\Box$ corresponds to Φ and the location $\ell_\Diamond$ corresponds to $\top U_{[1,2]} b$. One can check that this automaton follows strictly the above definition. Observe the edge labeled by b from $\ell_\Diamond$, without target state: it depicts the fact that $\delta(\top U b, b) = \top \vee (\top U b) = \top$. Intuitively, when the automaton has a copy in location '$\ell_\Diamond$' and reads a b, the copy can be *removed* (no condition

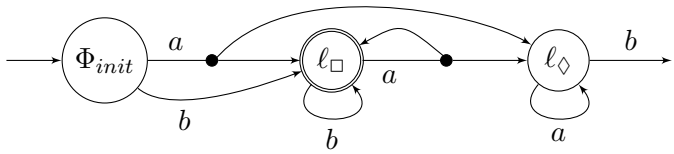


Figure 2.12: OCATA $\mathcal{A}_\Phi$ with $\Phi \equiv \Box(a \Rightarrow \Diamond b)$.

remains on this copy to satisfy Φ).
In the present example, Φ_{init} might be suppressed and $\ell_\Box$ used as initial location instead. However, in general, Φ_{init} is useful. For instance, when considering $\Phi' \equiv \Diamond a \wedge \Phi$, a conjunctive arc starts from location Φ_{init} and leads to a location representing $\Diamond a$ *and* to a location representing Φ, in way they are *both* verified.

2.6 Timed automata

This subsection is dedicated to timed automata. They have been introduced by Alur and Dill in [3] to model real-time systems and studied in details in [4]. This model is really popular. It is used in the academic sector as well as in the industry. It is in particular used by the standard model-checker UPPAAL [41]. We here define the syntax and the languages accepted by timed automata (over finite timed words) and by Büchi timed automata (over infinite timed words).

We start by defining the syntax and semantics of timed automata, based on the notion of *guard*. Then, we distinguish timed automata that accept finite timed words from Büchi timed automata, accepting infinite timed words, and define their languages.

Definition 2.64. *Let X be a set of clocks. We define the set of guards over X, denoted $\mathcal{G}(X)$, by the following grammar:*

$$\Phi := \top \quad | \quad x \bowtie c \quad | \quad \Phi_1 \wedge \Phi_2,$$

where $x \in X$, $c \in \mathbb{N}$ *and* $\bowtie \in \{<, \leqslant, >, \geqslant\}$. $x \bowtie c$ *is simply called a* clock constraint.

In the sequel, we will use some shortcuts to express guards over a clock x. For instance, for $n, m \in \mathbb{N}$, we will note $x = n$ instead of $(x \leqslant n) \wedge (x \geqslant n)$, and $x \in [n, m[$ instead of $(x \geqslant n) \wedge (x < m)$.

Definition 2.65. *A timed automaton (TA) (or a Büchi timed automaton (BTA), when interpreted over infinite words) is a tuple* $\mathcal{B} = (\Sigma, L, \ell_0, X, F, \delta)$, *where* Σ *is a finite* alphabet, L *is a finite set of* locations, $\ell_0 \in L$ *is the* initial location, X *is a finite set of* clocks, $F \subseteq L$ *is the set of* accepting locations, *and* $\delta \subseteq L \times \Sigma \times \mathcal{G}(X) \times 2^X \times L$ *is a finite set of* transitions. *For a transition* $(\ell, \sigma, g, r, \ell')$, *we say that* g *is its* guard, *and* r *its* reset.

Example 2.66. As an example, consider the TA $\mathcal{B}$ in Figure 2.13, over the alphabet $\Sigma = \{a, b\}$. $\mathcal{B}$ has two *locations* ℓ_0 and ℓ_1, such that ℓ_0 is initial and final. $\mathcal{B}$ has a unique clock x and its transition function is given by: $\delta = \{(\ell_0, b, \top, \varnothing, \ell_0), (\ell_0, a, \top, \{x\}, \ell_1), (\ell_1, b, x = 1, \varnothing, \ell_0)\}$.

To define what a *run* of a timed automaton is, we first define the notion of *valuation*, the notion of *configuration* and what are *timed* and *discrete successors* between configurations.

Definition 2.67. *Let* X *be a set of clocks.*

- *A* valuation *over* X *is a mapping* $v : X \to \mathbb{R}^+$.
- *The satisfaction of a clock constraint* $g \in \mathcal{G}(X)$ *by a valuation* v *over* X *is defined inductively in the usual way and denoted by* $v \models g$.
- *For* $t \in \mathbb{R}^+$, *we let* $v + t$ *to be the valuation defined by* $(v + t)(x) = v(x) + t$ *for all* $x \in X$.
- *For* $R \subseteq X$, *we let* $v[R := 0]$ *to be the valuation defined by* $(v[R := 0])(x) = 0$ *if* $x \in R$, *and* $(v[R := 0])(x) = v(x)$ *otherwise.*

Definition 2.68. *Let $\mathcal{B} = (\Sigma, L, \ell_0, X, F, \delta)$ be a (Büchi) timed automaton. A* configuration *of $\mathcal{B}$ is a pair (ℓ, v), where $\ell \in L$ and v is a valuation of the clocks in X. A configuration (ℓ, v) is* accepting *iff $\ell \in F$. We denote by* $\mathrm{Config}(\mathcal{B})$ *the set of all configurations of $\mathcal{B}$.*

Definition 2.69. *Let $\mathcal{B}$ be a (Büchi) timed automaton and (ℓ, v) be a configuration of $\mathcal{B}$.*

- *For all $t \in \mathbb{R}^+$, we have (time successor) $(\ell, v) \overset{t}{\rightsquigarrow} (\ell', v')$ iff $\ell = \ell'$ and $v' = v + t$.*
- *For all $\sigma \in \Sigma$, we have (discrete successor) $(\ell, v) \xrightarrow{\sigma} (\ell', v')$ iff there is $(\ell, \sigma, g, r, \ell') \in \delta$ such that $v \models g$ and $v' = v[r := 0]$.*

We write $(\ell, v) \xrightarrow{t,\sigma} (\ell', v')$ iff there is $(\ell'', v'') \in \mathrm{Config}(\mathcal{B})$ such that $(\ell, v) \overset{t}{\rightsquigarrow} (\ell'', v'') \xrightarrow{\sigma} (\ell', v')$.

As we will see thanks to the following definitions, a timed automaton is an acceptor of timed words.

Definition 2.70. *Let $\mathcal{B} = (\Sigma, L, \ell_0, X, F, \delta)$ be a (Büchi) timed automaton and $\theta = (\overline{\sigma}, \overline{\tau})$ be a (finite or infinite) timed word over Σ with $\overline{\sigma} = \sigma_1\sigma_2\sigma_3\ldots$ and $\overline{\tau} = \tau_1\tau_2\tau_3\ldots$. Let us note $t_i = \tau_i - \tau_{i-1}$ for all $1 \leqslant i \leqslant |\theta|$, assuming $\tau_0 = 0$. A* run *of $\mathcal{B}$ on θ is a (finite or infinite) sequence of time and discrete successor steps that is labelled by θ, i.e. a sequence of the form:*

$$(\ell_0, v_0) \overset{t_1}{\rightsquigarrow} (\ell_1, v_1) \xrightarrow{\sigma_1} (\ell_2, v_2) \overset{t_2}{\rightsquigarrow} (\ell_3, v_3) \xrightarrow{\sigma_2} \ldots$$
$$\overset{t_i}{\rightsquigarrow} (\ell_{2n-1}, v_{2n-1}) \xrightarrow{\sigma_i} (\ell_{2n}, v_{2n}) \ldots,$$

where v_0 assigns 0 to all clocks.
A finite run is accepting *if its last configuration is accepting. An infinite run is* accepting *iff there are infinitely many (ℓ_i, v_i) with $\ell_i \in F$ (i.e. we consider a Büchi acceptance condition).*
We say that a timed word θ is accepted *by $\mathcal{B}$ iff there is an accepting run of $\mathcal{B}$ on θ.*

In the sequel, we will often use examples in which a (Büchi) timed automaton $\mathcal{B}$ has only one clock. For the sake of simplicity, when it is the case, instead of denoting a run of $\mathcal{B}$ by

$$(\ell_0, v_0) \overset{t_1}{\rightsquigarrow} (\ell_1, v_1) \xrightarrow{\sigma_1} (\ell_2, v_2) \overset{t_2}{\rightsquigarrow} (\ell_3, v_3) \xrightarrow{\sigma_2} \dots$$
$$\overset{t_i}{\rightsquigarrow} (\ell_{2n-1}, v_{2n-1}) \xrightarrow{\sigma_i} (\ell_{2n}, v_{2n}) \dots,$$

and then give the value of each $v_i(x)$ for $1 \leqslant i \leqslant 2n$, we replace each v_i by the value of $v_i(x)$, giving the simpler notation

$$(\ell_0, v_0(x)) \overset{t_1}{\rightsquigarrow} (\ell_1, v_1(x)) \xrightarrow{\sigma_1} (\ell_2, v_2(x)) \overset{t_2}{\rightsquigarrow} (\ell_3, v_3(x)) \xrightarrow{\sigma_2} \dots$$
$$\overset{t_i}{\rightsquigarrow} (\ell_{2n-1}, v_{2n-1}(x)) \xrightarrow{\sigma_i} (\ell_{2n}, v_{2n}(x)) \dots$$

for this run.

Definition 2.71. *We denote by $L(\mathcal{B})$ the set of finite timed words accepted by the timed automaton $\mathcal{B}$ and by $L^\omega(\mathcal{B})$ the set of infinite timed words accepted by $\mathcal{B}$. We call $L(\mathcal{B})$ and $L^\omega(\mathcal{B})$ the* languages *of $\mathcal{B}$, respectively over finite and infinite timed words.*

Another view on the semantics of $\mathcal{B} = (\Sigma, B, \ell_0, X, F, \delta)$ is to associate it with a timed transition system $\mathsf{TTS}\,(\mathcal{B})$.

Definition 2.72. *Let $\mathcal{B} = (\Sigma, L, \ell_0, X, F, \delta)$ be a (Büchi) timed automaton. We define the timed transition system $\mathsf{TTS}\,(\mathcal{B}) = (\Sigma, S^{\mathsf{TTS}}, s_0^{\mathsf{TTS}}, \rightarrow^{\mathsf{TTS}}, \rightsquigarrow^{\mathsf{TTS}})$, where:*

- $S^{\mathsf{TTS}} = \mathrm{Config}\,(\mathcal{B})$ *is the set of states;*
- $s_0^{\mathsf{TTS}} = (\ell_0, v_0)$, *where v_0 is the valuation such that $v_0(x) = 0$ for all $x \in X$, is the initial state;*
- $\rightarrow^{\mathsf{TTS}} \subseteq S^{\mathsf{TTS}} \times \Sigma \times S^{\mathsf{TTS}}$ *is the discrete transition relation where* $((\ell, v), \sigma, (\ell', v')) \in \rightarrow^{\mathsf{TTS}}$ *iff* $(\ell, v) \xrightarrow{\sigma} (\ell', v')$;
- $\rightsquigarrow^{\mathsf{TTS}} \subseteq S^{\mathsf{TTS}} \times \mathbb{R}^+ \times S^{\mathsf{TTS}}$ *is the timed transition relation where* $((\ell, v), t, (\ell', v')) \in \rightsquigarrow^{\mathsf{TTS}}$ *iff* $(\ell, v) \overset{t}{\rightsquigarrow} (\ell', v')$.

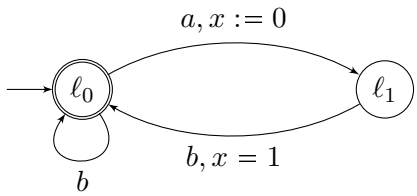


Figure 2.13: TA $\mathcal{B}$

We say that a finite timed word $\theta = (\overline{\sigma}, \overline{\tau})$ is accepted by $\mathsf{TTS}(\mathcal{B})$ iff there is in $\mathsf{TTS}(\mathcal{B})$ a finite path starting in s_0^{TTS}, ending in an accepting configuration and labeled by the finite sequence $\tau_1, \sigma_1, (\tau_2 - \tau_1), \sigma_2, \cdots, (\tau_n - \tau_{n-1}), \sigma_n$.
Similarly, an infinite timed word $\theta = (\overline{\sigma}, \overline{\tau})$ is accepted by $\mathsf{TTS}(\mathcal{B})$ iff there is in $\mathsf{TTS}(\mathcal{B})$ an infinite path starting in s_0^{TTS}, containing infinitely many $(\ell_i, v_i) \in \mathrm{Config}(\mathcal{B})$ with $\ell_i \in F$ and labelled by the infinite sequence $\tau_1, \sigma_1, (\tau_2 - \tau_1), \sigma_2, \cdots, (\tau_n - \tau_{n-1}), \sigma_n \cdots$.
We call the language of $\mathsf{TTS}(\mathcal{B})$ over finite words, denoted by $L(\mathsf{TTS}(\mathcal{B}))$, the set of finite timed words accepted by $\mathsf{TTS}(\mathcal{B})$. Similarly, we call the language of $\mathsf{TTS}(\mathcal{B})$ over infinite words, denoted by $L^\omega(\mathsf{TTS}(\mathcal{B}))$, the set of infinite timed words accepted by $\mathsf{TTS}(\mathcal{B})$.

Clearly, $\theta \in L(\mathcal{B})$ iff $\theta \in L(\mathsf{TTS}(\mathcal{B}))$ and $\theta \in L^\omega(\mathcal{B})$ iff $\theta \in L^\omega(\mathsf{TTS}(\mathcal{B}))$.

Example 2.73. Consider again the (Büchi) TA $\mathcal{B}$ in Figure 2.13. Let us observe the finite timed word $\theta = (a, 2)(b, 3)(b, 3.4)(a, 3.7)(b, 4.7)$. Here is an accepting run of $\mathcal{B}$ on θ. As previously explained, as $\mathcal{B}$ has only one clock, each v_i of Definition 2.70 has been replaced by the value of $v_i(x)$:

$$(\ell_0, 0) \overset{2}{\rightsquigarrow} (\ell_0, 2) \xrightarrow{a} (\ell_1, 0) \overset{1}{\rightsquigarrow} (\ell_1, 1) \xrightarrow{b} (\ell_0, 1) \overset{0.4}{\rightsquigarrow} (\ell_0, 1.4)$$
$$\xrightarrow{b} (\ell_0, 1.4) \overset{0.3}{\rightsquigarrow} (\ell_0, 1.7) \xrightarrow{a} (\ell_1, 0) \overset{1}{\rightsquigarrow} (\ell_1, 1) \xrightarrow{b} (\ell_0, 1).$$

Considering the infinite timed word

$$\theta = (a, 2)(b, 3)(b, 3.4)(a, 3.7)(b, 4.7)(a, 5)(b, 6)(a, 7)(b, 8) \dots ,$$

one can easily continue this run as an infinite accepting run. In fact, it is easy to see that:

$$L(\mathcal{B}) = \{\theta = (\overline{\sigma}, \overline{\tau}) \mid \sigma_{|\theta|} = b \text{ and } \forall 0 \leqslant i \leqslant |\theta| : \\ ((\sigma_i = a) \Rightarrow (\sigma_{i+1} = b \wedge \tau_{i+1} = \tau_i + 1))\}$$
$$\text{and } L^\omega(\mathcal{B}) = \{\theta = (\overline{\sigma}, \overline{\tau}) \mid \forall i \geqslant 0 : (\sigma_i = a) \Rightarrow (\sigma_{i+1} = b \wedge \tau_{i+1} = \tau_i + 1)\}.$$

As in the untimed setting, a desirable property of timed automata is the *determinism*. This notion is defined in such a way that, for each timed word θ, a deterministic timed automaton has at most one run on θ (as for the untimed case).

Definition 2.74. *Let $\mathcal{B} = (\Sigma, L, \ell_0, X, F, \delta)$ be a timed automaton or a Büchi timed automaton. $\mathcal{B}$ is deterministic iff for all $(\ell, \sigma, g, r, \ell_1)$ and $(\ell, \sigma', g', r', \ell_2)$ in δ, $\sigma = \sigma'$ implies that $g \wedge g'$ is* not *satisfiable.*

Example 2.75. The automaton of Figure 2.14 is not deterministic because of the elements of δ $(\ell_0, a, \top, \varnothing, \ell_0)$ and $(\ell_0, a, \top, \{x\}, \ell_1)$, starting from the initial location and holding the same letter: a. However, the automaton $\mathcal{B}$ of Figure 2.13 is deterministic: there are no different arcs starting from a same location and carrying a same letter.

Definition 2.76. *A timed automaton $\mathcal{B} = (\Sigma, L, \ell_0, X, F, \delta)$ is said* determinizable *over finite words if there is an automaton $\mathcal{B}_{det}$ such that $L(\mathcal{B}) = L(\mathcal{B}_{det})$. A Büchi timed automaton $\mathcal{B} = (\Sigma, L, \ell_0, X, F, \delta)$ is said* determinizable *over infinite words if there is a Büchi automaton $\mathcal{B}_{det}$ such that $L^\omega(\mathcal{B}) = L^\omega(\mathcal{B}_{det})$.*

Definition 2.77. *The* determinizability *problem asks, given a timed automaton (respectively, a Büchi timed automaton) if it is determinizable.*

Here are two theorems noticing that the determinization of timed automata is problematic.

Theorem 2.78 ([4]). *In general, timed automata and Büchi timed automata are not determinizable.*

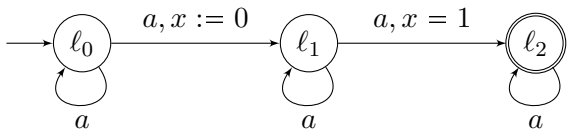


Figure 2.14: A non-determinizable timed automaton.

Example 2.79. It is well-known that the timed automaton of Figure 2.14 is not determinizable ([4]). Its language is the set of timed words containing two a's at a distance of 1 time unit. This timed automaton *guesses* which a will be followed by another a 1 time unit later: the transition from ℓ_0 to ℓ_1 is taken when reading this a. Intuitively, to represent this language by a deterministic timed automaton, we should distinguish this *guessed* a from others. This could be done resetting one different clock for each a read, and verifying which clock has value 1 while another a is read. Nevertheless, if n a's are read within 1 time unit, n different clocks are necessary, so that an unbounded number of clocks would be necessary (even in the setting of finite words).

The result given by the following theorem still surpass that of Theorem 2.78. It says that, in general, being given a timed automaton, it is not possible to decide if this automaton is determinizable.

Theorem 2.80 ([33]). *Over finite words, the problem of determinizability is undecidable for timed automata.*

Despite those negative result, subclasses of timed automata which are determinizable have been identified. As an example, in [12], a game-based algorithm tries to produce, for a given timed automaton, a deterministic timed automaton recognizing the same language. For non-determinizable timed automata, this algorithm produces a deterministic timed automaton recognizing an over-approximation of the original language.

Contrary to untimed automata, it is not possible in general to *complement* a (Büchi) timed automaton.

Definition 2.81. *Let $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$ be a timed automaton (respectively, be a Büchi timed automaton). We say that $\mathcal{B}$ is complementable if there is a timed automaton (respectively, a Büchi timed automaton) $\mathcal{B}^C$ such that $L(\mathcal{B}^C) = T\Sigma^\star \backslash L(\mathcal{B})$ (respectively, such that $L(\mathcal{B}^C) = T\Sigma^\omega \backslash L(\mathcal{B})$).*

Theorem 2.82 ([4]). *The classes of timed automata and Büchi timed automata are* not *closed by complement.*

Example 2.83. Let us observe again the automaton of Figure 2.14. The complement of its language, i.e. the set of timed words such that there is no pair of a's at a distance of 1 time unit, cannot be represented by any (Büchi) timed automaton. The intuition is the same as that why the automaton is not determinizable. Indeed, to determine the complement of this language, we should start a clock for each a read, to verify that there is no a exactly one time unit later. As un unbounded number of a's could be read within 1 time unit, an unbounded number of clocks would be necessary (even in the setting of finite words).

We now discuss the universality problem for timed automata.

Definition 2.84. *The universality problem asks, given a timed automaton (respectively, a Büchi timed automaton) $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$, if $L(\mathcal{B}) = T\Sigma^\star$ (respectively, if $L^\omega(\mathcal{B}) = T\Sigma^\omega$).*

While the universality problem is undecidable for Büchi timed automata and even for Büchi timed automata with a unique clock, it becomes decidable for timed automata with one clock over finite words.

Theorem 2.85 ([4]). *The universality problem is undecidable for timed automata and Büchi timed automata with at least two clocks.*

Theorem 2.86 ([48]). *Over finite words, the universality problem is decidable for timed automata with only one clock.*

Theorem 2.87 ([42]). *The universality problem is undecidable for Büchi timed automata with only one clock.*

2.7 Alternating timed automata

Let us now explain the notion of (one clock) *alternating timed automaton* (OCATA for short) ([51]), an extension of timed automata and alternating automata. Alternating timed automata extend timed automata in the same way alternating automata extend (Büchi) automata: they admit *conjunctive transitions* as well as disjunctive ones.

When using a conjunctive transition, an OCATA can create several copies of itself (including a copy of its clock !) that run in parallel and must *all* accept the suffix of the timed word. For example, Figure 2.15 displays an OCATA. Observe that the *arc* starting from ℓ_0 has two destinations: ℓ_0 and ℓ_1. When the automaton is in ℓ_0 with clock valuation $v \leqslant 2$, and reads an a, it spawns two copies of itself: the first reads the suffix of the word from (ℓ_0, v), and the latter from $(\ell_1, 0)$ (observe that the clock is reset on the branch to ℓ_1).
The *standard* semantics for OCATA [42, 49] is defined as an infinite transition system whose *configurations* are finite sets of pairs (ℓ, v), where ℓ is a location and v is the valuation of the (unique) clock. Intuitively, each configuration thus represents the current state of all the copies of the (unique!) clock that run in parallel in the OCATA. The transition system is *infinite* because one cannot bound, a priori, the number of different clock valuations that can appear in a single configuration, thereby requiring peculiar techniques, such as well quasi orderings (see [51]) to analyse it.

To formally define an OCATA, we let $\Gamma(L)$ be the set of formulas defined by the following grammar:

$$\gamma := \top \quad | \quad \bot \quad | \quad \gamma_1 \vee \gamma_2 \quad | \quad \gamma_1 \wedge \gamma_2 \quad | \quad \ell \quad | \quad x \bowtie c \quad | \quad x.\gamma$$

where $c \in \mathbb{N}$, $\bowtie \in \{<, \leqslant, >, \geqslant\}$ and $\ell \in L$. We recall that $x \bowtie c$ is in fact a *clock constraint*. Intuitively, the expression $x.\gamma$ means that clock x must be reset to 0.

Definition 2.88 ([51]). *A **one-clock alternating timed automaton** (OCATA)*

is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ *where* Σ *is a finite alphabet,* L *is a finite set of locations,* ℓ_0 *is the initial location,* $F \subseteq L$ *is a set of accepting locations,* $\delta : L \times \Sigma \to \Gamma(L)$ *is the transition function.*

To simplify the use of the transition function, we explain how $\delta(\ell, \sigma)$ can always be written in disjunctive normal form.

Remark that, for all γ_1, γ_2 in $\Gamma(L)$: $x.(\gamma_1 \vee \gamma_2) = x.\gamma_1 \vee x.\gamma_2$, $x.(\gamma_1 \wedge \gamma_2) = x.\gamma_1 \wedge x.\gamma_2$, $x.x.\gamma = x.\gamma$, $x.(x \bowtie c) = 0 \bowtie c$, $x.\top = \top$ and $x.\bot = \bot$. Thus, we can write any formula of $\Gamma(L)$ in disjunctive normal form, and, from now on, we assume that $\delta(\ell, \sigma)$ is written in this form. That is, for all ℓ, σ, we have $\delta(\ell, \sigma) = \bigvee_j \bigwedge_k A_{j,k}$, where each term $A_{j,k}$ is of the form ℓ, $x.\ell$, $x \bowtie c$ or $0 \bowtie c$, with $\ell \in L$ and $c \in \mathbb{N}$. We call *arc* of the OCATA $\mathcal{A}$ a triple $(\ell, \sigma, \bigwedge_k A_{j,k})$ such that $\bigwedge_k A_{j,k}$ is a disjunct in $\delta(\ell, \sigma)$.

Example 2.89. As an example, consider the OCATA $\mathcal{A}$ in Figure 2.15, over the alphabet $\Sigma = \{a\}$. $\mathcal{A}$ has three *locations* ℓ_0, ℓ_1 and ℓ_2, such that ℓ_0 is initial and ℓ_0 and ℓ_1 are final. The transition function of $\mathcal{A}$ is given by: $\delta(\ell_0, a) = (\ell_0 \wedge x > 1) \vee (\ell_0 \wedge x.\ell_1 \wedge x \leqslant 2)$, $\delta(\ell_1, a) = (\ell_1 \wedge x \neq 1) \vee \ell_2$ and $\delta(\ell_2, a) = \ell_2$. The *arcs* of $\mathcal{A}$ are thus $(\ell_0, a, \ell_0 \wedge x > 1)$, $(\ell_0, a, \ell_0 \wedge x.\ell_1 \wedge x \leqslant 2)$, $(\ell_1, a, \ell_1 \wedge x \neq 1)$, (ℓ_1, a, ℓ_2) and (ℓ_2, a, ℓ_2). Observe that, in the figure, we represent the (conjunctive) arc $(\ell_0, a, \ell_0 \wedge x.\ell_1 \wedge x \leqslant 2)$ by an arrow carrying the clock constraint $x \leqslant 2$ and then splitting in two branches connected respectively to ℓ_0 and ℓ_1 (possibly with different resets: the reset of clock x is depicted by $x := 0$). Intuitively, *taking the arc* $(\ell_0, a, \ell_0 \wedge x.\ell_1 \wedge x \leqslant 2)$ means that, when reading an a from location ℓ_0 and clock value $v \leqslant 2$, the automaton should start *two copies of itself*, one in location ℓ_0, with clock value v, and a second in location ℓ_1 with clock value 0. Both copies should accept the suffix for the word to be accepted (this notion will be defined formally in a following paragraph).

Classical semantics. We will now explain the standard semantics of OCATA [51]. We start by defining what is a *state* and a *configuration* of an OCATA and how it can move from one configuration to another using the notion of *minimal model*.

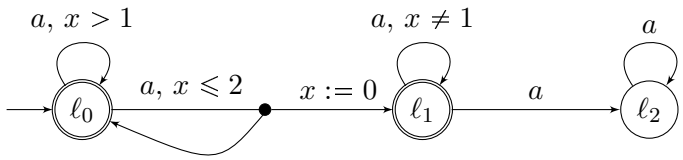


Figure 2.15: OCATA $\mathcal{A}$

Definition 2.90. *We call* state *of an OCATA* $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ *a couple* (ℓ, v) *where* $\ell \in L$ *and* v *is a valuation of its unique clock. A state* (ℓ, v) *is* accepting *iff* $\ell \in F$.
We note $S = L \times \mathbb{R}^+$ *the state space of* $\mathcal{A}$.

Definition 2.91. *A* configuration *of an OCATA* $\mathcal{A}$ *is a (possibly empty) finite set of states of* $\mathcal{A}$. *The* initial configuration *of* $\mathcal{A}$ *is* $\{(\ell_0, 0)\}$. *A configuration is* accepting *iff all the states it contains are accepting (in particular, the empty configuration is accepting).*
For a configuration C *and a delay* $t \in \mathbb{R}^+$, *we note* $C + t$ *the configuration* $\{(\ell, I + t) | (\ell, I) \in C\}$.
We note $\mathrm{Config}\,(\mathcal{A})$ *the set of all configurations of* $\mathcal{A}$.

We now define the satisfaction relation '$\models_v$' enabling to express the notion of *minimal model.*

Definition 2.92. *Let* $M \in \mathrm{Config}\,(\mathcal{A})$ *be a configuration of an OCATA* $\mathcal{A}$, *and* $v \in \mathbb{R}^+$. *We define the satisfaction relation '*$\models_v$*' on* $\Gamma(L)$ *as:*

- $M \models_v \top$,
- $M \models_v \gamma_1 \wedge \gamma_2$ *iff* $M \models_v \gamma_1$ *and* $M \models_v \gamma_2$,
- $M \models_v \gamma_1 \vee \gamma_2$ *iff* $M \models_v \gamma_1$ *or* $M \models_v \gamma_2$,
- $M \models_v \ell$ *iff* $(\ell, v) \in M$,
- $M \models_v x \bowtie c$ *iff* $v \bowtie c$,

- $M \models_v x.\gamma$ *iff* $M \models_0 \gamma$.

Definition 2.93. *We say that M is a minimal model of the formula $\gamma \in \Gamma(L)$ with respect to $v \in \mathbb{R}^+$ iff $M \models_v \gamma$ and there is no $M' \subsetneq M$ such that $M' \models_v \gamma$.*

Remark that a formula γ can admit several minimal models (at most one for each disjunct in the case of a formula of the form $\gamma = \bigvee_j \bigwedge_k A_{j,k}$, see Examples 2.94 and 2.95). Intuitively, for $\ell \in L, \sigma \in \Sigma$ and $v \in \mathbb{R}^+$, each minimal model of $\delta(\ell, \sigma)$ with respect to v is one of the configurations the automaton can reach from the state (ℓ, v) by reading σ.

Example 2.94. Let us consider again the OCATA of Figure 2.15. A minimal model M of $\delta(\ell_0, a)$ with respect to 1.7 must be such that: $M \models_{1.7} (\ell_0 \wedge x > 1) \vee (\ell_0 \wedge x.\ell_1 \wedge x \leqslant 2)$. As $1.7 > 1$ and $1.7 \leqslant 2$, we have:

$$
\begin{array}{llll}
 & M \models_{1.7} (\ \ell_0 \wedge x > 1\ ) & \vee & (\ \ell_0 \wedge x.\ell_1 \wedge x \leqslant 2\ ) \\
\text{iff} & M \models_{1.7} \ell_0 & \text{or} & (\ M \models_{1.7} \ell_0 \text{ and } M \models_{1.7} x.\ell_1\ ) \\
\text{iff} & (\ell_0, 1.7) \in M & \text{or} & (\ (\ell_0, 1.7) \in M \text{ and } M \models_0 \ell_1\ ) \\
\text{iff} & (\ell_0, 1.7) \in M & \text{or} & (\ (\ell_0, 1.7) \in M \text{ and } (\ell_1, 0) \in M\ ).
\end{array}
$$

So, $\{(\ell_0, 1.7)\}$ and $\{(\ell_0, 1.7), (\ell_1, 0)\}$ are two models of $\delta(\ell_0, a)$ with respect to 1.7, but as $\{(\ell_0, 1.7)\} \subseteq \{(\ell_0, 1.7), (\ell_1, 0)\}$, $\{(\ell_0, 1.7)\}$ is the unique *minimal model* of $\delta(\ell_0, a)$ with respect to 1.7.

Example 2.95. Let us consider the OCATA of Figure 2.16. A minimal model M of $\delta(\ell_0, a)$ with respect to 0.2 must be such that: $M \models_{0.2} (\ell_0 \wedge x.\ell_1) \vee (\ell_0 \wedge$

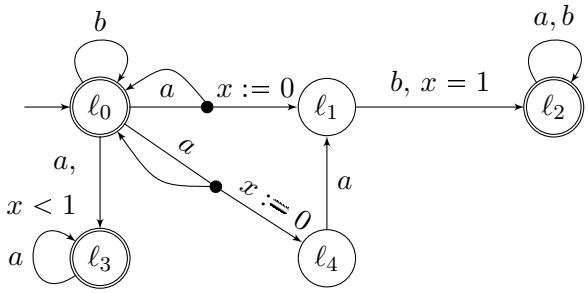


Figure 2.16: The OCATA $\mathcal{A}$

$x.\ell_4) \vee (\ell_3 \wedge x < 1)$. As $0.2 < 1$ we have:

	$M \models_{0.2} (\ \ell_0 \wedge x.\ell_1\ )$	$\vee$	$(\ \ell_0 \wedge x.\ell_4\ )$
		$\vee$	$(\ \ell_3 \wedge x < 1\ )$
iff	$(\ M \models_{0.2} \ell_0$ and $M \models_{0.2} x.\ell_1\ )$	or	$(\ M \models_{0.2} \ell_0$ and $M \models_{0.2} x.\ell_4\ )$
		or	$(\ M \models_{0.2} \ell_3\ )$
iff	$(\ (\ell_0, 0.2) \in M$ and $M \models_0 \ell_1\ )$	or	$(\ (\ell_0, 0.2) \in M$ and $M \models_0 \ell_4\ )$
		or	$(\ (\ell_3, 0.2) \in M\ )$
iff	$(\ (\ell_0, 0.2) \in M$ and $(\ell_1, 0) \in M\ )$	or	$(\ (\ell_0, 0.2) \in M$ and $(\ell_4, 0) \in M\ )$
		or	$(\ (\ell_3, 0.2) \in M\ )$

So, $\{(\ell_0, 0.2), (\ell_1, 0)\}$, $\{(\ell_0, 0.2), (\ell_4, 0)\}$ and $\{(\ell_3, 0.2)\}$ are three minimal models of $\delta(\ell_0, a)$ with respect to 0.2.

We now define some notations enabling to simply express what are the possible *successors* of a given configuration of an OCATA.

Definition 2.96. *Let $\mathcal{A}$ be an OCATA $\mathcal{A}$ and (ℓ, v) be one of its states. We let* $\mathsf{Succ}((\ell, v), \sigma) := \{M \mid M$ *is a minimal model of* $\delta(\ell, \sigma)$ *with respect to* $v\}$. *We lift the definition of* Succ *to configurations C as follows:* $\mathsf{Succ}(C, \sigma)$ *is the set of all configurations C' of the form $\cup_{s \in C} M_s$, where, for all $s \in C$: $M_s \in \mathsf{Succ}(s, \sigma)$. That is, each $C' \in \mathsf{Succ}(C, \sigma)$ is obtained by choosing one minimal model M_s in* $\mathsf{Succ}(s, \sigma)$ *for each $s \in C$, and taking the union of all those M_s.*

Example 2.97. Let us consider the state $(\ell_0, 0.2)$ of the OCATA of Figure 2.15. $\mathsf{Succ}((\ell_0, 0.2), a) = \{M \mid M \text{ is a minimal model of } \delta(\ell_0, a) \text{ with respect to } 0.2\}$. A minimal model M of $\delta(\ell_0, a)$ with respect to 0.2 must be such that: $M \models_{0.2} (\ell_0 \wedge x > 1) \vee (\ell_0 \wedge x.\ell_1 \wedge x \leqslant 2)$. As $0.2 \not> 1$ and $0.2 \leqslant 2$, we have:

$$\begin{array}{ll} & M \models_{0.2} (\ \ell_0 \wedge x > 1\ ) \ \vee\ (\ \ell_0 \wedge x.\ell_1 \wedge x \leqslant 2\ ) \\ \text{iff} & M \models_{0.2} (\ \ell_0 \wedge x.\ell_1 \wedge x \leqslant 2\ ) \\ \text{iff} & M \models_{0.2} \ell_0 \text{ and } M \models_{0.2} x.\ell_1 \\ \text{iff} & (\ell_0, 0.2) \in M \text{ and } M \models_0 \ell_1 \\ \text{iff} & (\ell_0, 0.2) \in M \text{ and } (\ell_1, 0) \in M. \end{array}$$

Hence, $C = \{(\ell_0, 0.2), (\ell_1, 0)\}$ is the unique minimal model of $\delta(\ell_0, a)$ with respect to 0.2: $\mathsf{Succ}((\ell_0, 0.2), a) = \{C\}$.
In a similar way, let us determine $\mathsf{Succ}(C, a)$. We know that the unique minimal models of $\delta(\ell_0, a)$ with respect to 0.2 is C, and need those of $\delta(\ell_1, a)$ with respect to 0. A minimal model M of $\delta(\ell_1, a)$ with respect to 0 must be such that: $M \models_0 (\ell_1 \wedge x \neq 1) \vee \ell_2$. As $0 \neq 1$, we have:

$$\begin{array}{lll} & M \models_0 (\ \ell_1 \wedge x \neq 1\ ) & \vee \quad \ell_2 \\ \text{iff} & M \models_0 \ell_1 & \text{or} \quad M \models_0 \ell_2 \\ \text{iff} & (\ell_1, 0) \in M & \text{or} \quad (\ell_2, 0) \in M. \end{array}$$

So, $\{(\ell_1, 0)\}$ and $\{(\ell_2, 0)\}$ are two minimal models of $\delta(\ell_1, a)$ with respect to 0. Hence, $\mathsf{Succ}(C, a)$ has two elements: $C \cup \{(\ell_1, 0)\}$ and $C \cup \{(\ell_2, 0)\}$: $\mathsf{Succ}(C, a) = \{\{(\ell_0, 0.2), (\ell_1, 0)\}, \{(\ell_0, 0.2), (\ell_1, 0), (\ell_2, 0)\}\}$.

Runs of OCATA. We can now formally define the notion of *run* of an OCATA in the classical semantics. Each new configuration in the run is obtained in two steps: letting time elapse and performing a discrete step. These steps characterize the *semantics* of an OCATA, on which is based the definition of *run*.

Definition 2.98. *Let $\mathcal{A}$ be an OCATA. The* semantics of $\mathcal{A}$ *is the transition system* $\mathsf{TTS}(\mathcal{A}) = (\mathrm{Config}(\mathcal{A}), \rightsquigarrow, \longrightarrow)$ *on configurations of $\mathcal{A}$ defined as follows:*

- *the transition relation* $\rightsquigarrow$ *takes care of the elapsing of time:* $\forall t \in \mathbb{R}^+$, $C \overset{t}{\rightsquigarrow} C'$ *iff* $C' = C + t$. *We let* $\rightsquigarrow = \bigcup_{t \in \mathbb{R}^+} \overset{t}{\rightsquigarrow}$.
- *the transition relation* $\longrightarrow$ *takes care of discrete transitions between locations:* $C \xrightarrow{\sigma} C'$ *iff* $C' \in \mathsf{Succ}(C, \sigma)$. *We let* $\longrightarrow = \bigcup_{\sigma \in \Sigma} \xrightarrow{\sigma}$.

Definition 2.99. *Let* $\mathcal{A}$ *be an OCATA of state space* S. *Let* $\theta = (\overline{\sigma}, \overline{\tau})$ *be a (finite or infinite) timed word and let us note* $t_i = \tau_i - \tau_{i-1}$ *for all* $1 \leqslant i \leqslant |\theta|$, *assuming* $\tau_0 = 0$. *A* run *of* $\mathcal{A}$ *on* θ *is a finite or infinite sequence of discrete and continuous transitions in* $\mathsf{TTS}(\mathcal{A})$ *that is labelled by* θ, *i.e. a sequence of the form:* $C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \ldots \overset{t_n}{\rightsquigarrow} C_{2n-1} \xrightarrow{\sigma_n} C_{2n} \ldots$.

In the rest of this thesis, we (sometimes) use the abbreviation $C_i \xrightarrow{t,\sigma} C_{i+2}$ for $C_i \overset{t}{\rightsquigarrow} C_{i+1} = C_i + t \xrightarrow{\sigma} C_{i+2}$.

As in the untimed setting, for all pairs of configurations C, C' such that $C' \in \mathsf{Succ}(C + t, \sigma)$ for some t and σ, each $s \in C$ can be associated with a unique set $\mathsf{dest}(C, C', s) \subseteq C'$ containing all the 'successors' of s in C' and obtained as follows. By definition, $C' = \bigcup_{\overline{s} \in C} M_{\overline{s}}$, where each $M_{\overline{s}} \in \mathsf{Succ}(\overline{s}, \sigma)$ is the minimal model that has been chosen for $\overline{s}$ when computing $\mathsf{Succ}(C + t, \sigma)$. Then, $\mathsf{dest}(C, C', s) = M_s$.

The function dest allows to define a DAG representation of runs, as is usual with alternating automata. We regard a run π as a rooted DAG $G_\pi = (V, \rightarrow)$, whose vertices V correspond to the states of the OCATA (vertices at depth i correspond to C_{2i}), and whose set of edges $\rightarrow$ expresses the OCATA transitions. Here is its formal definition.

Definition 2.100. *Let* $\mathcal{A}$ *be an OCATA and* $\pi = C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \ldots \overset{t_n}{\rightsquigarrow} C_{2n-1} \xrightarrow{\sigma_n} C_{2n} \ldots$ *be a run of* $\mathcal{A}$. *We define the rooted DAG* $G_\pi = (V, \rightarrow)$ *with:*

- $V = \bigcup_{0 \leqslant i \leqslant |\theta|} V_i$, *where for all* $0 \leqslant i \leqslant |\theta|$: $V_i = \{(s, i) \mid s \in C_{2i}\}$ *is the set of all vertices of depth* i *;*

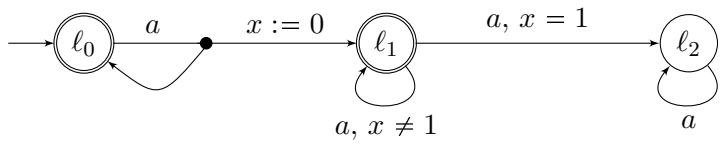


Figure 2.17: OCATA $\mathcal{A}$

- *the root of* G_π *is* $((\ell_0, 0), 0)$ *; and*
- $(s_1, i_1) \to (s_2, i_2)$ *iff* $i_2 = i_1 + 1$ *and* $s_2 \in \mathsf{dest}(C_{2i_1}, C_{2i_2}, s_1)$.

Example 2.101. Let us consider the OCATA $\mathcal{A}$ of Figure 2.17. Figure 2.18 displays the two possible representations of the run prefix π of $\mathcal{A}$ on the word $(a, 0.1)(a, 0.2)(a, 0.9)(a, 1.2)\ldots$: grey boxes highlight the successive configurations. π shows why the number of clock copies cannot be bounded in general: if $\mathcal{A}$ reads a word containing n a's between instants 0 and 1, n copies of the clock are created in location ℓ_1.
The edges of the DAG of the run are obtained thanks to function dest. For instance, the edges from states of C_2 to states in C_4 are obtained thanks to $\mathsf{dest}(C_2, C_4, (\ell_0, 0.1))$ and $\mathsf{dest}(C_2, C_4, (\ell_1, 0))$. As C_4 is computed from $C_2 + 0.1 = \{(\ell_0, 0.2), (\ell_1, 0.1)\}$ reading an a, it consists in the minimal models of $\delta(\ell_0, a)$ with respect to 0.2, and of $\delta(\ell_1, a)$ with respect to 0.1. In fact, $C_4 = \bigcup_{s \in C_2} M_s$, for $M_{(\ell_0, 0.1)} = \{(\ell_0, 0.2), (\ell_1, 0)\}$ and $M_{(\ell_1, 0)} = \{(\ell_1, 0.1)\}$. Hence, $\mathsf{dest}(C_2, C_4, (\ell_0, 0.1)) = M_{(\ell_0, 0.1)} = \{(\ell_0, 0.2), (\ell_1, 0)\}$ and there is an edge from state $(\ell_0, 0.1)$ in C_2 to states $(\ell_0, 0.2)$ and $(\ell_1, 0)$ in C_4. In a similar way, $\mathsf{dest}(C_2, C_4, (\ell_1, 0)) = M_{(\ell_1, 0)} = \{(\ell_1, 0.1)\}$: there is an edge from state $(\ell_1, 0)$ in C_2 to state $(\ell_1, 0.1)$ in C_4.

Language of OCATA. We can now define the accepted language of an OCATA, over finite and infinite timed words. We will see that OCATA define timed languages.

For finite timed words, the characterisation of runs given by Definition 2.99 is more convenient:

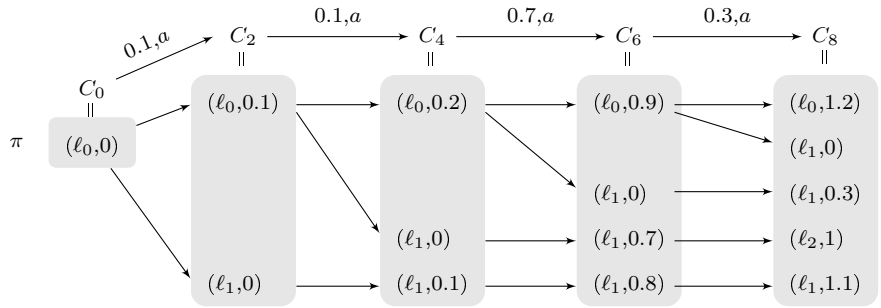


Figure 2.18: Two views on a finite run of the OCATA in Figure 2.17.

Definition 2.102. *A finite run is* accepting *iff its last configuration C_{2n} is accepting and we say that a finite timed word is accepted by $\mathcal{A}$ iff there exists an accepting finite run of $\mathcal{A}$ on this word.*
We note $L(\mathcal{A})$ the language of all finite timed words accepted by $\mathcal{A}$.

Nevertheless, as in the untimed case, to define when an infinite timed word is accepted by $\mathcal{A}$, we need to use the DAG characterisation of runs.

Definition 2.103. *We call* branch *of a run represented by a DAG G_π a (finite or infinite) path in G_π. We note $Bran^\omega(G_\pi)$ the set of all* infinite *branches of G_π and, for $\beta \in Bran^\omega(G_\pi)$, we note $Inf(\beta)$ the set of locations occurring infinitely often along β.*
A run represented by a DAG G_π is accepting *iff $\forall \beta \in Bran^\omega(G_\pi)$, $Inf(\beta) \cap F \neq \emptyset$ (i.e. we consider a Büchi acceptance condition). We say that an infinite timed word θ is accepted by $\mathcal{A}$ iff there exists an accepting run of $\mathcal{A}$ on θ.*
We note $L^\omega(\mathcal{A})$ the language of all infinite timed words accepted by $\mathcal{A}$.

Example 2.104. The run π of the OCATA $\mathcal{A}$ of Figure 2.17, on the infinite word $\theta = (a, 0.1)(a, 0.2)(a, 0.9)(a, 1.2)\ldots$ will not be accepting, for any suffix of θ, because of the branch of π leading to $(\ell_2, 1)$: it is easy to see that there is no possible way to leave the non-accepting location ℓ_2 of $\mathcal{A}$. In the same way, the run π on the finite word $(a, 0.1)(a, 0.2)(a, 0.9)(a, 1.2)$ is not accepting because C_8 is not accepting. In contrary, the run of $\mathcal{A}$ on $(a, 0.1)(a, 0.2)(a, 0.9)$

is $C_0 \xrightarrow{0.1,a} C_2 \xrightarrow{0.1,a} C_4 \xrightarrow{0.7,a} C_6$. It is accepting because C_6 is accepting. It is not difficult to see that:

$$L(\mathcal{A}) = \{\theta = (\overline{\sigma}, \overline{\tau}) \mid \forall 1 \leqslant i \leqslant |\theta|,\ \forall i < j \leqslant |\theta|,\ \tau_j \neq \tau_i + 1\}, \text{ and}$$

$$L^\omega(\mathcal{A}) = \{\theta = (\overline{\sigma}, \overline{\tau}) \mid \forall i \geqslant 1,\ \forall j > i,\ \tau_j \neq \tau_i + 1\}.$$

Classical results on OCATA. We here present decidability results for classical problems on OCATA studied in the literature.

Let us first consider the *emptiness problem*. Remark that the emptiness problem is *complementary* to the *universality problem*. We will deduce from Theorem 2.85 (claiming that the universality problem is undecidable for timed automata) that the emptiness problem is undecidable for alternating timed automata, over finite words. Indeed, the complement of the language (over finite words !) of a timed automaton can easily be represented by an alternating timed automaton in which we 'dualised' the transition relation (as formally defined below) and exchange accepting and non-accepting locations. Its exact definition is given hereunder.

We first define the dual of a formula $\Phi \in \Gamma(L)$.

Definition 2.105. *The dual of a formula $\Phi \in \Gamma(L)$ is the formula $\overline{\Phi}$ defined inductively as follows.*

- $\overline{false} = true$ *and* $\overline{true} = false$;
- $\forall \ell \in L,\ \overline{\ell} = \ell$;
- $\overline{\Phi_1 \vee \Phi_2} = \overline{\Phi_1} \wedge \overline{\Phi_2}$;
- $\overline{\Phi_1 \wedge \Phi_2} = \overline{\Phi_1} \vee \overline{\Phi_2}$;
- $\overline{x.\Phi} = x.\overline{\Phi}$;

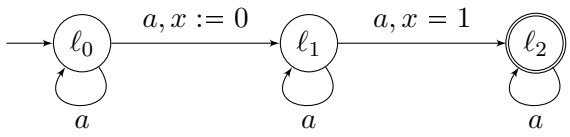


Figure 2.19: The timed automaton $\mathcal{B}$.

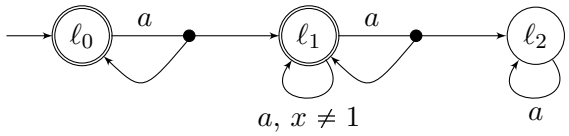


Figure 2.20: The alternating timed automaton $\mathcal{B}^C$.

- *the dual of a clock constraint is its negation (for example:* $\overline{x \leqslant c} = x > c$*).*

Definition 2.106. *Let* $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$ *be a timed automaton over finite words. We define* $\mathcal{B}^C$ *to be the alternating timed automaton given by the tuple* $(\Sigma, L, \ell_0, L \backslash F, \overline{\delta})$*, where* $\overline{\delta}(\ell, a) = \overline{\delta(\ell, a)}$*, for all* $\ell \in L$ *and* $a \in \Sigma$*.*

Theorem 2.107. *Let* $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$ *be a timed automaton over finite words. Then,* $L(\mathcal{B}^C) = T\Sigma^\star \backslash L(\mathcal{B})$*.*

Example 2.108. Let us observe the timed automaton $\mathcal{B} = (\Sigma, L, \ell_0, F, \delta)$ of Figure 2.19. We have already seen that the complement of its language, i.e. the set of timed words such that there is no pair of a's at a distance of 1 time unit, cannot be represented by any timed automaton (see Example 2.83). However, the alternating timed automaton $\mathcal{B}^C$ is such that $L(\mathcal{B}^C) = T\Sigma^\star \backslash L(\mathcal{B})$. It is represented in Figure 2.20. Its transition function is obtained as follows :

- $\overline{\delta}(\ell_0, a) = \overline{\ell_0 \vee x.\ell_1} = \ell_0 \wedge x.\ell_1$,
- $\overline{\delta}(\ell_1, a) = \overline{\ell_1 \vee (x = 1 \wedge \ell_2)} = \ell_1 \wedge (x \neq 1 \vee \ell_2) = (\ell_1 \wedge x \neq 1) \vee (\ell_1 \wedge \ell_2)$,
- $\overline{\delta}(\ell_2, a) = \overline{\ell_2} = \ell_2$.

Hence, as a direct consequence of Theorem 2.85, we obtain the following one:

Theorem 2.109. *Over finite words, the emptiness problem is undecidable for alternating timed automata (with more than one clock).*

Now, let us consider an alternating timed automaton over finite words. It is possible to algorithmically complete it into an alternating timed automaton over infinite words accepting a word $\theta(\sigma)^\omega$ iff θ was accepted by this automaton over finite words. As the language of the basis automaton is empty iff that of the deformed automaton is empty, we obtain the following theorem:

Theorem 2.110. *Over infinite words, the emptiness problem is undecidable for alternating timed automata (with more than one clock).*

The following theorem states that the decidability of emptiness is recovered for OCATA (that only own *one* clock), over finite words. In part 4.2, we present techniques to perform model-checking over finite words using OCATA. Then, we extend these techniques and results to the setting of infinite words. Observing the previous and following theorems, one can imagine that this passage from finite to infinite words is not trivial: working on infinite words, we approach the decidability frontier of OCATA.

Theorem 2.111 ([51]). *Over finite words, the emptiness problem is decidable for OCATA.*

In [52], Parys and Walukiewicz defined a weak version of OCATA, in the sense that the acceptance condition is a *weak parity condition*. For this kind of OCATA, interpreted over infinite words, they show that the emptiness problem is decidable.
Their OCATA are defined as tuples $\mathcal{A} = (\Sigma, L, \ell_0, \Omega, \delta)$ where $\Omega : L \to \mathbb{N}$ determines the weak parity acceptance condition. A run G_π of $\mathcal{A}$ is accepted if, for all branches $\beta \in Bran^\omega(G_\pi)$, with $\beta = ((\ell_0, v_0), 0)((\ell_1, v_1), 1)((\ell_2, v_2), 2) \dots ((\ell_n, v_n), n) \dots$, we have that: $\min\{\Omega(\ell_i) \mid i \in \mathbb{N}\}$ is even.
In their work, they consider timed words whose time sequences are strictly increasing and unbounded, i.e. timed words of the form $\theta = (\overline{\sigma}, \overline{\tau})$ such that $\forall i \geqslant 1$,

$\tau_i < \tau_{i+1}$ and $\forall t \in \mathbb{R}^+$, $\exists i \geqslant 1$ such that $\tau_i \geqslant t$: they call such words *non Zeno timed words*. In [52], they prove the following theorem:

Theorem 2.112 ([52]). *It is decidable whether a given OCATA $\mathcal{A} = (\Sigma, L, \ell_0, \Omega, \delta)$, with $\Omega : L \to \{0, 1\}$, accepts some non Zeno timed word. The complexity of the problem is non-primitive recursive.*

Let us now discuss the complementation of OCATA. In general, the complement of an alternating timed automaton with a Büchi acceptance condition (such as those considered in this thesis) is an alternating timed automaton with a *co-Büchi acceptance condition*.
An alternating timed automaton with *co-Büchi* acceptance condition $\mathcal{C} = (\Sigma, L, \ell_0, F, \delta)$ has an acceptance condition which is *complementary* to that of a Büchi alternating timed automaton: it is the unique difference between these two kind of automata. In fact, a run of $\mathcal{C}$ (over an infinite word) is now accepting if it visits only a *finite* number of times the locations of F.
The complement of a Büchi alternating timed automaton $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ is a *co-Büchi* alternating timed automaton in which we dualised the transition relation of $\mathcal{A}$: its exact definition is given hereunder. Intuitively, we obtain the complement of the original language dualizing the transition function and *complementing* the acceptance condition. Nevertheless, the complement of a Büchi acceptance condition is not a Büchi acceptance condition. If we only exchange accepting and non-accepting locations of $\mathcal{A}$ and keep a Büchi acceptance condition, it means we want that $L \backslash F$ is visited infinitely often, which is not complementary to the fact that F is visited infinitely often: a run might visit both F and $L \backslash F$ infinitely often. The right complementary of a Büchi acceptance condition on F is to visit F only finitely often: it is a co-Büchi acceptance condition on F.

As far as we know, no simple method enables to complement an alternating timed automaton with Büchi acceptance condition by giving an automaton of this same class.

Definition 2.113. *Let $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ be a Büchi alternating timed automa-*

ton. We define $\mathcal{A}^C$ to be the co-Büchi alternating timed automaton given by the tuple $(\Sigma, L, \ell_0, F, \overline{\delta})$, where $\overline{\delta}(\ell, a) = \overline{\delta(\ell, a)}$, for all $\ell \in L$ and $a \in \Sigma$.

In the sequel, for an alternating timed automaton $\mathcal{A}$, we use the notation $L^{\omega}_{co-B}(\mathcal{A})$ to denote its language when considering a co-Büchi acceptance condition.

Theorem 2.114. *Let $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ be a Büchi alternating timed automaton. Then, $L^{\omega}_{co-B}(\mathcal{A}^C) = T\Sigma^{\omega} \backslash L^{\omega}(\mathcal{A})$.*

Example 2.115. Let us observe the Büchi alternating timed automaton $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ of Figure 2.21. Its language is the set of timed words in which each a is followed by a b from 1 to 2 units of time later. The co-Büchi alternating timed automaton $\mathcal{A}^C$ is such that $L^{\omega}_{co-B}(\mathcal{A}^C) = T\Sigma^{\omega} \backslash L^{\omega}(\mathcal{A})$. It is represented in Figure 2.22. Its transition function is obtained as follows :

- $\overline{\delta}(\ell_0, a) = \overline{\ell_0 \wedge x.\ell_1} = \ell_0 \vee x.\ell_1$,
- $\overline{\delta}(\ell_0, b) = \overline{\ell_0} = \ell_0$,
- $\overline{\delta}(\ell_1, a) = \overline{\ell_1} = \ell_1$,
- $\overline{\delta}(\ell_1, b) = \overline{\ell_1 \vee (\ell_2 \wedge x \in [1,2])} = \ell_1 \wedge (\ell_2 \vee x \notin [1,2]) = (\ell_1 \wedge \ell_2) \vee (\ell_1 \wedge x \notin [1,2])$,
- $\overline{\delta}(\ell_2, a) = \overline{\ell_2} = \ell_2$,
- $\overline{\delta}(\ell_2, b) = \overline{\ell_2} = \ell_2$.

2.8 Metric Temporal Logic

In this section, we recall the definition of Metric Temporal Logic (MTL): it is a real-time extension of LTL, that has been proposed by Koymans ([39]), in

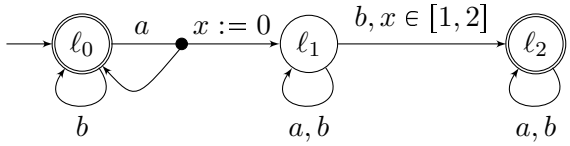


Figure 2.21: The Büchi alternating timed automaton $\mathcal{A}$.

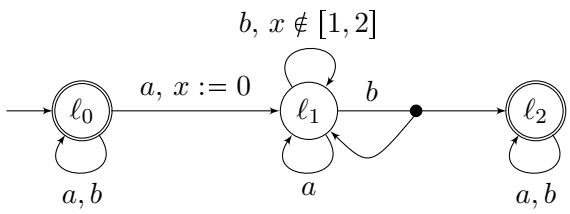


Figure 2.22: The co-Büchi alternating timed automaton $\mathcal{A}^C$.

which the until modality is labelled by an interval. We begin by defining its syntax and its semantics. We then define different problems using this logic we are interested in.

2.8.1 MTL syntax and pointwise semantics

First, let us formally define the syntax of MTL.

Definition 2.116 (MTL syntax)**.** *Given a finite alphabet Σ, the formulas of MTL are defined by the following syntax, where $\sigma \in \Sigma$, $I \in \mathcal{I}(\mathbb{N}^\infty)$:*

$$\Phi := \top \quad | \quad \sigma \quad | \quad \Phi_1 \wedge \Phi_2 \quad | \quad \neg\Phi \quad | \quad \Phi_1 U_I \Phi_2.$$

In the sequel, we rely on the following usual shortcuts. The 'eventually' operator: $\Diamond_I \Phi$ stands for $\top U_I \Phi$. The 'always' operator: $\Box_I \Phi$ stands for $\neg\Diamond_I\neg\Phi$. We will also use $\Box\Phi$ for $\Box_{[0,\infty[}\Phi$, $\Diamond\Phi$ for $\Diamond_{[0,\infty[}\Phi$, and $\Phi_1 \tilde{U}_I \Phi_2$ for $\neg(\neg\Phi_1 U_I \neg\Phi_2)$.

Definition 2.117. *Let Φ be an MTL formula, we note $Sub(\Phi)$ the set of all subformulas of Φ, i.e.:*

- $Sub(\Phi) = \{\Phi\}$ *when* $\Phi \in \{\top\} \cup \Sigma$,
- $Sub(\neg\Phi) = \{\neg\Phi\} \cup Sub(\Phi)$ *and*
- $Sub(\Phi) = \{\Phi\} \cup Sub(\Phi_1) \cup Sub(\Phi_2)$ *when* $\Phi \equiv \Phi_1 U_I \Phi_2$ *or* $\Phi \equiv \Phi_1 \wedge \Phi_2$.

We define the *size* of an MTL formula as follows:

Definition 2.118. *Let Φ be an MTL formula. We let $|\Phi|$ denote the* size of Φ, *defined as the* number *of U or $\tilde{U}$ modalities it contains.*

We are now able to define the *pointwise semantics* of MTL. We will see that this semantics is adapted for the setting of finite words as well as for that of infinite words.

Definition 2.119. *Given a timed word $\theta = (\overline{\sigma}, \overline{\tau})$ over Σ, a position $1 \leqslant i \leqslant |\theta|$ and an MTL formula Φ, we define that θ satisfies Φ from position i, written $(\theta, i) \models \Phi$, using the following rules:*

1. $(\theta, i) \models \top$,
2. $(\theta, i) \models \sigma$ *iff* $\sigma_i = \sigma$,
3. $(\theta, i) \models \Phi_1 \wedge \Phi_2$ *iff* $(\theta, i) \models \Phi_1$ *and* $(\theta, i) \models \Phi_2$,
4. $(\theta, i) \models \neg\Phi$ *iff* $(\theta, i) \not\models \Phi$,
5. $(\theta, i) \models \Phi_1 U_I \Phi_2$ *iff* $\exists i \leqslant j \leqslant |\theta|$, *such that* $(\theta, j) \models \Phi_2, \tau_j - \tau_i \in I$ *and* $\forall i \leqslant k < j, (\theta, k) \models \Phi_1$.

We say that θ satisfies *Φ, written $\theta \models \Phi$, iff $(\theta, 1) \models \Phi$. We note $[\![\Phi]\!] = \{\theta \mid |\theta| < \infty$ and $\theta \models \Phi\}$ the* finite timed language *of Φ and $[\![\Phi]\!]^\omega = \{\theta \mid |\theta| = +\infty$ and $\theta \models \Phi\}$ the* infinite timed language *of Φ.*

The semantics for the U operator we have just given is known as the *unstrict* semantics. Although we mostly use this semantics in this thesis, in the literature, a *strict* semantics is also often used for the U operator:

6. $(\theta, i) \models \Phi_1 U_I \Phi_2$ iff $\exists i{<}j \leqslant |\theta|$, such that $(\theta, j) \models \Phi_2, \tau_j - \tau_i \in I$ and $\forall i{<}k < j, (\theta, k) \models \Phi_1$.

We will sometimes use the strict semantics and then specify it.

Example 2.120. MTL enables to express *quantitative* properties, on the sequence of events. For instance, we can express the fact that '*every occurrence of p is followed by an occurrence of q between 2 and 3 time units later*' by the MTL formula: $\Box(p \Rightarrow \Diamond_{[2,3]} q)$.

2.8.2 The problems

In this section, we define the problems on MTL we are interested in. We also recall classical results concerning them.

Satisfiability. The first main problem on the MTL logic consists in verifying that there exists a timed word satisfying a given MTL formula.

Definition 2.121. *Let Φ be an MTL formula. We say that Φ is satisfiable over finite (respectively infinite) timed words if there exists a finite (respectively infinite) timed word $\overline{\sigma}$ such that $\overline{\sigma}$ satisfies Φ.*
The satisfiability problem over finite (respectively infinite) timed words is the problem of deciding if a given MTL formula Φ is satisfiable over finite (respectively infinite) timed words.

Model-Checking. The second main problem on the MTL logic we are interested in is the 'MTL Model-Checking problem'. Being given an automaton or a Büchi automaton $\mathcal{B}$ and an MTL formula Φ, it consists in verifying whether the language

of $\mathcal{B}$ is included in the language of Φ. In other words, it enables to verify if all accepting runs of $\mathcal{B}$ satisfy the property described by Φ.

Definition 2.122. *Let $\mathcal{B}$ be an automaton and Φ be an MTL formula. The MTL model-checking problem over finite timed words is to decide if the following inclusion holds: $L(\mathcal{A}) \subseteq [\![\Phi]\!]$.*
Identically, let $\mathcal{C}$ be a Büchi automaton and Φ be an MTL formula. The MTL model-checking problem over infinite timed words is to decide if the following inclusion holds: $L^\omega(\mathcal{A}) \subseteq [\![\Phi]\!]^\omega$.

A first paper of Henzinger [7] stated the undecidability of these two versions of the MTL model-checking problem. Indeed, the proof of [7] also stated that two other versions of the MTL model-checking problem were undecidable: the MTL model-checking over finite and infinite words for the *continuous semantics* (which will be defined in a following section). However, in [51], Ouaknine and Worrell proved that the MTL model-checking problem is decidable on the pointwise semantics, over finite words. This created a need for new irreproachable proofs for the decidability or undecidability of all other versions of the MTL model-checking problem. In the following, we review the key ideas of these reviewed proofs *on the pointwise semantics*. They led to the following result:

Theorem 2.123 ([50, 51])**.** *On the pointwise semantics, the MTL model-checking problem over finite words is non primitive recursive, while the MTL model-checking problem over infinite words is undecidable.*

To simplify the key intuitions presented in the sequel, we will consider the strict semantics of the U operator, as a reminder:

$$(\theta, i) \models \Phi_1 U_I \Phi_2 \text{ iff } \exists i{<}j \leqslant |\theta|, \text{ such that } (\theta, j) \models \Phi_2, \tau_j - \tau_i \in I \text{ and}$$

$$\forall i{<}k < j, (\theta, k) \models \Phi_1$$

The proof may be adapted for the unstrict semantics of the U operator, using

atomic propositions instead of simple letters[1].

The ideas of proof presented in this section are also explained in [16], while the original proofs can be found in [50] and [51].

The decidability result over finite words comes from the fact that, with an MTL formula, we cannot capture the behaviours of a perfect channel machine but only of a channel machine *with insertion errors*.

Let us consider a perfect channel machine $\mathcal{C} = (S, s_0, M, C, \Delta)$, where S is a finite set of locations, $s_0 \in S$ is the initial location, M is a finite set of messages, C is a finite set of FIFO channels and $\Delta \subseteq S \times A \times S$ is the transition relation over the set of actions $A = \{c!m, c?m \mid c \in C \text{ and } m \in M\}$. Let us note $C = \{c_1, \ldots, c_{|C|}\}$. A configuration of $\mathcal{C}$ is a tuple $(s, w_1, \ldots, w_{|C|})$ where $s \in S$ and, for all $1 \leqslant i \leqslant |C|$, w_i is a word representing the content of channel c_i. A run of $\mathcal{C}$ in a sequence of steps consisting in the reading of an action of A: for all $c_i \in C$ and $m \in M$, we have a step

- $(s, w_1, \ldots, w_{|C|}) \xrightarrow{c_i!m} (s', w'_1, \ldots, w'_{|C|})$ iff $(s, c_i!m, s') \in \Delta$, $w'_i = w_i m$ and for all $j \neq i$, $w'_j = w_j$;
- $(s, w_1, \ldots, w_{|C|}) \xrightarrow{c_i?m} (s', w'_1, \ldots, w'_{|C|})$ iff $(s, c_i?m, s') \in \Delta$, $w_i = m w'_i$ and for all $j \neq i$, $w'_j = w_j$.

For a channel machine with insertion errors, this last kind of steps is replaced by the following one:

- $(s, w_1, \ldots, w_{|C|}) \xrightarrow{c_i?m} (s', w'_1, \ldots, w'_{|C|})$ iff $(s, c_i?m, s') \in \Delta$, $w_i = m w'_i$ *or* $w_i = w'_i$, and for all $j \neq i$, $w'_j = w_j$.

[1] The use of atomic propositions instead of simple letters is not really important as we are considering an *undecidable* problem.

In the case $w_i = w'_i$, we implicitely assume that m has been inserted on the channel c_i, so that it can be read from it.

We first show an idea of why we cannot encode runs of a perfect channel machine by an MTL formula, following the ideas of [16]. Then, we show that we can however encode by MTL formulas the runs of a channel machine with insertion error. Let us recall that the halting problem of a channel machine with insertion errors is non primitive recursive over finite words. As it is straightforward, thanks to this encoding, to reduce this problem to the MTL model-checking problem over finite words, we will conclude that the MTL model-checking problem over finite words is at least non primitive recursive.

A run of the machine could be encoded by a timed word, described by an MTL formula, in the following way.

1. The first letter of the timed word is the initial location of the machine:

$$\Phi_{init} = s_0.$$

2. To represent the evolution between locations of the machine when it is reading a word, the letters forming the timed word alternate between locations of S and actions of A:

$$\Phi_{alt} \equiv \Box \left(\bigwedge_{s \in S} \left(s \Rightarrow \left(\bot U \bigvee_{a \in A} a \right) \right) \wedge \bigwedge_{a \in A} \left(a \Rightarrow \left(\bot U \bigvee_{s \in S} s \right) \right) \right).$$

3. Each triple (s, a, s') of consecutive letters of the timed word starting with a location $s \in S$ must be a triple of Δ:

$$\Phi_{\Delta} \equiv \Box \left(\bigwedge_{s \in S} \left(s \Rightarrow \left(\bigvee_{(s,a,s') \in \Delta} \left(\bot U \left(a \wedge \left(\bot U s' \right) \right) \right) \right) \right) \right).$$

4. To force the channels to be FIFO, we furthermore require (i) that each action of type $c!m$ is followed exactly one time unit later by the corresponding

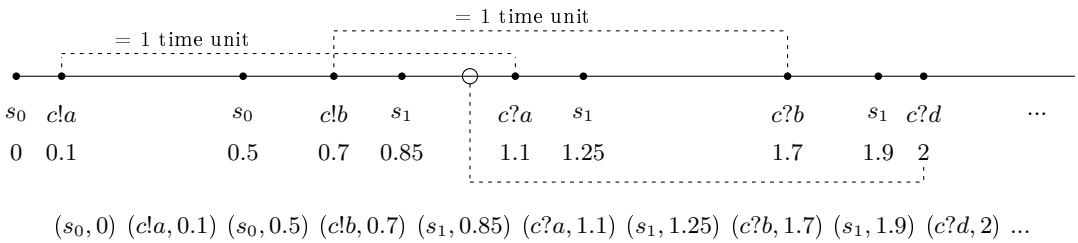


Figure 2.23: Encoding of a run presenting an insertion error of $\mathcal{C}$

action $c?m$ and (ii) that each action of type $c?m$ is preceded exactly one time unit earlier by the corresponding action $c!m$. Unfortunately, only the (i) part of this requirement can be express by an MTL formula:

$$\Phi_{FIFO1} \equiv \Box \left(\bigwedge_{c!m \in A} \left(c!m \Rightarrow \Diamond_{[1,1]} c?m \right) \right).$$

One could think that we can express (ii) by the MTL formula

$$\Phi_{FIFO2} \equiv \Box \left(\bigwedge_{c?m \in A} \left(\left(\Diamond_{[1,1]} c?m \right) \Rightarrow c!m \right) \right).$$

However, the timed word '$(c!m, 0.1)(c!m, 0.3)(c?m, 3)$' satisfies Φ_{FIFO2} because no letter is read one time unit before the $c?m$. Indeed, observing the semantics of MTL, we see we have no condition to check what occurs at time 2 since there is no event occuring at this time. So, the meaning of Φ_{FIFO2} would be that 'each $c?m$ is preceded one time earlier, either by a $c!m$ or by no letter'. In fact, no MTL formula (on the pointwise semantics) enables to express the desired property: we cannot prevent a channel machine to make insertion errors.

Example 2.124. Figure 2.23 gives the encoding of a run of $\mathcal{C}$ (presented in Figure 2.24), presenting an insertion error, as a timed word.

The halting problem of a channel machine with insertion errors is non primitive recursive over finite words, and it is straightforward to reduce this problem

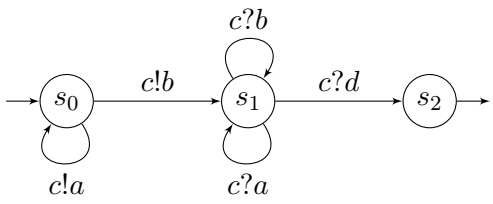


Figure 2.24: Channel machine $\mathcal{C}$

to the MTL model-checking problem (on the pointwise semantics) over finite words. Hence, we can only conclude that this problem is at least non primitive recursive. In [51], Ouaknine and Worrell present a non primitive recursive algorithm to decide this MTL model-checking problem. Their technique is based on the construction, for a given MTL formula Φ, of an OCATA $\mathcal{A}_\Phi$ recognizing $[\![\Phi]\!]$ (this construction will be detailed in a following section and then used in our contributions). However, they prove in [50] that the MTL model-checking problem (on the pointwise semantics) is undecidable over infinite words. They obtain this result studying a special type of channel machine called 'insertion channel machines with emptiness testing'. They first prove that the recurrent-state problem is undecidable for such machines. Then, they show this problem can be reduced to the MTL model-checking problem over infinite words (on the pointwise semantics).

Reactive synthesis. As for the LTL case, the MTL reactive synthesis aims to find how a system satisfying a given MTL property may be designed, against the constraints of the reality. The MTL reactive synthesis problem considers a property, given by an MTL formula, and an *environment*, representing the constraints of the reality in the conception of the desired system. This problem may be seen as a game in which a *controller* competes against an *environment*. At each round, the controller and the environment suggest an action and a delay: the player that proposes the smaller delay is authorized to play its action at the proposed delay. Hence, starting from the empty timed word and adding at each round the pair (σ, t) played, for a certain action σ and a certain delay t, a timed

word is created. The aim of the controller is to play following a strategy such that, however the environment plays, the produced timed words satisfy the MTL property.

Let us formally define this problem. We consider an alphabet $\Sigma = \Sigma_C \cup \Sigma_E$ partitioned into a set of controllable actions Σ_C and a set of environment actions Σ_E. Let us first define what is a *strategy* for the controller.

Definition 2.125. *A* strategy *for the controller is a mapping* $f\colon T\Sigma^\star \to (\Sigma_C \times \mathbb{R}^+) \uplus \{\bot\}$. *From the finite timed word* θ *already created,* $f(\theta)$ *proposes a pair* (σ, t)*, consisting in an action an a delay, or the special action* $\bot$*, denoting the fact that the controller does not want to perform any action.*

We may imagine that the environment proposes also such a strategy, though it is not authorized to perform $\bot$. The combination of both strategies produces a set of timed words as follows:

- we start with the empty timed word, and then,
- at each round, the controller and the environment propose (respectively) pairs (σ_C, d_C) and (σ_E, d_E) based on their strategies: the one with the smallest delay is performed, or the one of the environment if the controller proposes $\bot$;
- this pair is concatenated to the current play and a next round starts.

When both the controller and the environment propose the same delay, we consider both possibilities, so that this scheme produces a set of finite timed words, not a unique one. Instead of considering a strategy for the environment, in the sequel, we define the set of finite timed words *consistent* with the strategy f of the controller.

Definition 2.126. *The set of finite timed words* consistent *with the strategy* f *of the controller is the smallest set of timed words* $T\Sigma_f^\star$ *containing the empty timed word* ε*, and closed by the following operations. For all* $\theta \in T\Sigma_f^\star$*:*

- *if* $f(\theta) = (a,t) \in \Sigma_C \times \mathbb{R}^+$*:*
  we let $\theta \cdot (a, T+t) \in T\Sigma_f^\star$ *and* $\theta \cdot (b, T+t') \in T\Sigma_f^\star$ *for all* $t' \leqslant t$ *and* $b \in \Sigma_E$*,*

- *if* $f(\theta) = \bot$*:*
  we let $\theta \cdot (b, T+t') \in T\Sigma_f^\star$ *for all* $t' \in \mathbb{R}^+$ *and* $b \in \Sigma_E$*,*

where $T = 0$ *if* $\theta = \epsilon$ *and* $(c,T) \in \Sigma \times \mathbb{R}^+$ *is the last letter of* θ *otherwise.*

We are now able to define the *reactive synthesis problem.*

Definition 2.127. *The MTL* reactive synthesis problem *(RS) is to decide, given an MTL formula* Φ*, whether there exists a strategy* f *of the controller such that every non-empty timed word consistent with strategy* f *satisfies the formula* Φ*, i.e.* $T\Sigma_f^\star \subseteq [\![\Phi]\!] \cup \{\varepsilon\}$[2]*.*
A formula Φ *such that there exists such a strategy is said* realizable *for RS.*
If a formula is realizable, computing a strategy for this formula is called the synthesis *problem.*

Example 2.128. Let us consider the partitioned alphabet $\Sigma = \Sigma_C \cup \Sigma_E$, where $\Sigma_C = \{b\}$ and $\Sigma_E = \{a\}$. We also consider the MTL formula:

$$\Phi \;\equiv\; \Box\big((a \wedge \Diamond_{>0} a \wedge \Box_{]0,1]} \neg a) \;\Rightarrow\; (\Diamond_{=1} b)\big).$$

We define the following strategy f of the controller: for all $\theta \in T\Sigma^\star$, $f(\theta) = (b,1)$. $T\Sigma_f^\star$ contains the timed words θ_f such that, for each a in θ_f:

- either this is the last letter of θ_f,
- or there is another a at a distance smaller than 1 time unit from this,
- or there is another a at a distance strictly bigger than 1 time unit, but then, there is also a b exactly one time unit after this a.

[2] The empty word ε is only added for techical reasons: it is easier to recursively define $T\Sigma_f^\star$ when starting from ε

So, $T\Sigma_f^\star \subseteq [\![\Phi]\!] \cup \{\varepsilon\}$ and Φ is realizable for RS.
We notice that, for this first example, the strategy of the controller may be easily implemented. This could not be the case as witnesses the following instance. Let us now consider the MTL formula:

$$\Phi' \equiv \Box\big((a \wedge \Diamond_{>1} a) \Rightarrow (\Diamond_{=1} b)\big).$$

Φ' is also realizable for RS but the strategy f of the controller is more complex. This strategy must consist in *treating* all the a's played by the environment, by chronological order, as follows. When, in a certain instant t, an a played in time stamp τ_a is the one to treat, the controller must propose a pair $(b, \tau_a + 1 - t)$. However, the environment may play an unbounded number of a's between time stamps τ_a and $\tau_a + 1$ so that this strategy requires a unbounded memory, which is not implementable in practice.

Implementable and bounded reactive synthesis problems We here introduce a second concept of synthesis, more relevant for practical interests. Indeed, let Φ be an MTL formula: if Φ is realizable for RS, it is possible that the strategy for the synthesis is overly complex. Sometimes, it could not be implementable in practice, because it must record too much information about time, with an infinite precision for instance. In contrary, we would like to constrain the problem to only look into a subset of strategies presenting more regularity. We choose to opt for strategies representable by *symbolic transition systems*.

Let us first define what is a *symbolic alphabet* and a *symbolic transition system*.

Definition 2.129. *Let Σ be an alphabet and X be a set of clocks. A symbolic alphabet Γ based on (Σ, X) is a finite subset of $\Sigma \times \mathcal{G}(X) \times 2^X$.*

A symbolic word $\gamma = (\sigma_1, g_1, R_1) \cdots (\sigma_n, g_n, R_n)$ over Γ generates a set of timed words over Σ, denoted by $tw(\gamma)$. Symbolic action (σ, g, R) denotes the fact that action σ is performed when constraint g is satisfied, in which case clocks in R are reset. Here is the formal definition of $tw(\gamma)$.

Definition 2.130. *Let Γ be a symbolic alphabet based on (Σ, X) and $\gamma = (\sigma_1, g_1, R_1) \cdots (\sigma_n, g_n, R_n)$ be an element of $\Gamma^\star$. $\theta \in tw(\gamma)$ iff $\theta = (\sigma_1, \tau_1) \cdots (\sigma_n, \tau_n)$ and there is a sequence $(v_i)_{i=0}^n$ of valuations over X such that:*

- *v_0 is the valuation such that $v_0(x) = 0$, $\forall x \in X$, and*
- *$\forall 1 \leqslant i \leqslant n$, $(v_{i-1} + \tau_i - \tau_{i-1}) \models g_i$ and $v_i = (v_{i-1} + \tau_i - \tau_{i-1})[R_i := 0]$ (assuming $\tau_0 = 0$).*

Example 2.131. Let us consider the alphabet $\Sigma = \{a, b\}$ and the set of clocks $X = \{x\}$. Here is an example of a symbolic alphabet Γ based on (Σ, X):

$$\begin{aligned}\Gamma \ := \ \{ \ & (a, x = 0, \emptyset), (a, x = 0, \{x\}), (b, x = 0, \emptyset), (b, x = 0, \{x\}), \\ & (a, x = 1, \emptyset), (b, x > 1, \{x\}), (a, x \geqslant 1, \emptyset), (a, x < 2, \{x\}) \quad \}\end{aligned}$$

Here is a symbolic word γ over Γ:

$$\gamma \ := \ (a, x = 1, \emptyset)\,(a, x < 2, \{x\})\,(b, x = 0, \emptyset)\,(a, x \geqslant 1, \emptyset).$$

Let us consider the following timed word over Σ:

$$\theta \ := \ (a, 1)\,(a, 1.3)\,(b, 1.3)\,(a, 2.4).$$

We claim that $\theta \in tw(\gamma)$. Indeed, we consider the sequence $(v_i)_{i=0}^4$ of valuations over $X = \{x\}$ such that: $v_0(x) = 0$, $v_1(x) = 1$, $v_2 = 0$, $v_3 = 0$ and $v_4 = 1.1$. We have:

- v_0 is the valuation such that $v_0(x) = 0$, and
- $v_0 + 1 - 0 = 1 \models x = 1$, $v_1 + 1.3 - 1 = 1.3 \models x < 2$, $v_2 + 1.3 - 1.3 = 0 \models x = 0$ and $v_3 + 2.4 - 1.3 = 1.1 \models x \geqslant 1$. Moreover $v_1 = (v_0 + 1 - 0)$, $v_2 = (v_1 + 1.3 - 1)[\{x\} := 0]$, $v_3 = (v_2 + 1.3 - 1.3)$ and $v_4 = (v_3 + 2.4 - 1.3)$.

It is easy to see that, in fact:

$$tw(\gamma) \ := \ \{\, (a, 1)(a, v_1(x))(b, v_1(x))(a, v_2(x)) \,|\, 1 \leqslant v_1(x) < 2 \text{ and } v_2(x) \geqslant v_1(x) + 1 \,\}$$

Definition 2.132. *A* symbolic transition system *(STS) over a symbolic alphabet* Γ *based on* (Σ, X) *is a tuple* $\mathcal{ST} = (S, s_0, \Delta, F)$ *where:*

- S *is possibly infinite set of locations,*
- $s_0 \in S$ *is the initial location,*
- $\Delta \subseteq S \times \Gamma \times S$ *is the transition relation, and*
- $F \subseteq S$ *is a set of accepting locations*

In the sequel, when all the locations of an STS are accepting, we omit its last component, noting $\mathcal{ST} = (S, s_0, \Delta)$. An STS with finitely many locations may be seen as a simple *timed automaton* (see Definition 2.65). We so define a configuration of an STS in a same way as a configuration of a TA:

Definition 2.133. *Let* $\mathcal{ST} = (S, s_0, \Delta, F)$ *be an STS over a symbolic alphabet* Γ *based on* (Σ, X)*. A* configuration *of* $\mathcal{ST}$ *is a pair* (s, v)*, where* $s \in S$ *and* v *is a valuation of the clocks in* X*. A configuration* (s, v) *is* accepting *iff* $s \in F$*.*

Example 2.134. Let us consider the alphabet $\Sigma = \{a, b\}$, the set of clocks $X = \{x\}$ and the symbolic alphabet Γ based on (Σ, X).
The STS $\mathcal{ST} = (S, s_0, \Delta, F)$ (over Γ) of Figure 2.25 (left) is such that: $S = \{s_0, s_1\}$, $\Delta = \{(s_0, (a, \top, \{x\}), s_1), (s_0, (b, \top, \varnothing), s_0), (s_1, (b, x = 1, \varnothing), s_0)\}$ and $F = \{s_0\}$. It has a finite set of locations and may be seen has the timed automaton of Figure 2.25 (right).
Now, let us consider the STS $\mathcal{ST}' = (S', s'_0, \Delta', F')$ (over Γ) of Figure 2.26. It is such that: $S' = \{s'_0, s'_1, s'_2, s'_3, s'_4, \dots\}$ is its infinite set of locations, $\Delta = \{(s'_0, (a, \top, \{x\}), s'_1), (s'_0, (b, \top, \varnothing), s'_0), (s'_1, (b, x = 1, \varnothing), s'_0), (s'_1, (b, x \geqslant 1, \{x\}), s'_2), (s'_2, (b, x \geqslant 1, \{x\}), s'_3), (s'_3, (b, x \geqslant 1, \{x\}), s'_4), \dots\}$ and $F = \{s'_0, s'_2, s'_3, s'_4, \dots\}$. It cannot be considered as a timed automaton because of its *infinite* set of locations.

We now define the possible languages recognized by an STS. We start by defining what is a *path* of an STS.

Figure 2.25: The STS $\mathcal{ST} = (S, s_0, \Delta, F)$ (left) and its corresponding timed automaton (right).

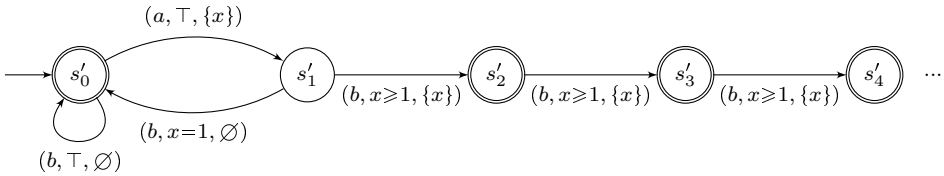


Figure 2.26: The STS $\mathcal{ST}' = (S', s'_0, \Delta', F')$.

Definition 2.135. *Let $\mathcal{ST}$ be an STS. $\pi = s_1 \xrightarrow{b_1} s_2 \xrightarrow{b_2} \cdots \xrightarrow{b_n} s_{n+1}$ is a* path *of $\mathcal{ST}$ iff, $\forall 1 \leqslant i \leqslant n$, $(s_i, b_i, s_{i+1}) \in \Delta$.*
Such a finite path is accepting iff it ends in an accepting location.
We call the symbolic word $b_1 b_2 \cdots b_n$ (over Γ) the trace *of π.*

At first sight, an STS is an acceptor of symbolic words over Γ.

Definition 2.136. *Let $\mathcal{ST}$ be an STS. The* symbolic language *of $\mathcal{ST}$, denoted by $L_{symb}(\mathcal{ST})$, is the set of finite symbolic words over Γ that are traces of finite accepting paths starting from the initial location s_0.*

However, an STS is also an acceptor of timed words over Σ.

Definition 2.137. *Let $\mathcal{ST}$ be an STS. The timed language accepted by $\mathcal{ST}$ is defined by $L(\mathcal{ST}) = tw(L_{symb}(\mathcal{ST}))$.*

Example 2.138. Let us consider again the alphabet $\Sigma = \{a, b\}$, the set of clocks $X = \{x\}$ and the symbolic alphabet Γ based on (Σ, X). We observe again the

STS $\mathcal{ST}$ of Figure 2.25 (left). The symbolic language of $\mathcal{ST}$ is:

$$\begin{aligned} L_{symb}(\mathcal{ST}) &= \{\gamma = \gamma_1\gamma_2\ldots\gamma_n \mid \big(\gamma_n = (b, x = 1, \varnothing) \text{ or } \gamma_n = (b, \top, \varnothing)\big) \text{ and} \\ &\quad \big(\forall 1 \leqslant i \leqslant n : \ \gamma_i = (a, \top, \{x\}) \ \Rightarrow \ \gamma_{i+1} = (b, x = 1, \varnothing)\big)\}. \end{aligned}$$

The timed language accepted by $\mathcal{ST}$ is also the language of the corresponding timed automaton. It is:

$$\begin{aligned} L(\mathcal{ST}) := tw(L_{symb}(\mathcal{ST})) &= \{\theta = (\overline{\sigma}, \overline{\tau}) \mid \sigma_{|\theta|} = b \text{ and } \forall 1 \leqslant i \leqslant |\theta| : \\ &\quad ((\sigma_i = a) \ \Rightarrow \ (\sigma_{i+1} = b \wedge \tau_{i+1} = \tau_i + 1))\} \end{aligned}$$

As for a classical timed automaton, we say that a STS $\mathcal{ST}$ is *deterministic* if there are no distinct transitions $(s, (a, g_1, R_1), s_1)$ and $(s, (a, g_2, R_2), s_2)$ in Δ with a valuation v verifying both g_1 and g_2. However, we can also define the interesting notion of *symbol-determinism* as follows.

Definition 2.139. *Let $\mathcal{ST}$ be a STS. $\mathcal{ST}$ is* symbol-deterministic *iff $(s, b, s_1) \in \Delta$ and $(s, b, s_2) \in \Delta$ imply $s_1 = s_2$.*

A symbol-deterministic STS associates to every symbolic word $\gamma \in \Gamma^\star$ at most one path starting from s_0 whose trace is γ. Whereas determinism implies symbol-determinism, the reverse implication is false.

Example 2.140. The STS $\mathcal{ST}$ of Figure 2.25 (left) is deterministic: all pairs of arcs from the same location are labelled by symbolic letters whose letter of Σ are different. In particular, it is also symbol-deterministic.
The STS $\mathcal{ST}'$ of Figure 2.26 is symbol-deterministic: all pairs of arcs from the same location are labelled by different symbolic letters. However, it is not a deterministic STS because of the two arcs $(s'_1, (b, x = 1, \varnothing), s'_0)$ and $(s'_1, (b, x \geqslant 1, \{x\}), s'_2)$. Indeed, the valuation v over $\{x\}$ such that $v(x) = 1$ is such that $v \models x = 1$ and $v \models x \geqslant 1$.

Remark 2.141. In a deterministic STS $\mathcal{ST} = (S, s_0, \Delta, S_f)$, every timed word θ can be read by at most one path. More formally, there exists at most one path π whose trace γ verifies $\theta \in tw(\gamma)$. In that case, we denote by $\Delta(s_0, \theta)$ the

unique (if it exists) pair (s, v) with s the last location of the path π, and v the valuation of clocks in X obtained by reading $\theta \in tw(\gamma)$ from valuation v_0 (as defined above).

To define a notion of reactive synthesis where the controller is characterized by an STS, we will lift the definition of $T\Sigma_f^\star$ (see Definition 2.126) for a strategy given as an STS. To do this, we still need to define what is an *enabled* timed action.

Definition 2.142. *Let $\mathcal{ST} = (S, s_0, \Delta, S_f)$ be an STS, $s \in S$ and v a valuation over the clocks of $\mathcal{ST}$. We say that a timed action $(a, t) \in \Sigma \times \mathbb{R}^+$ is* enabled *in $\mathcal{ST}$ at a configuration (s, v) if there exists a transition $(s, (a, g, R), s')$ in Δ such that $v + t \models g$.*
We note $\mathbf{Enabled}_{\mathcal{ST}}^{timed}((s, v))$ the set of timed actions enabled in $\mathcal{ST}$ at the configuration (s, v).

Example 2.143. Let us consider again the STS $\mathcal{ST}$ of Figure 2.25 (left). Let us observe the pair (s_1, v_1) with $v_1(x) = 0.2$. We have: $\mathbf{Enabled}_{\mathcal{ST}}^{timed}(s_1, v_1) = \{(b, 0.8)\}$. Indeed the unique transition from s_1 is $(s_1, (b, x = 1, \emptyset), s_0)$ and the unique t such that $v + t \models x = 1$ is $t = 0.8$.
If we consider the pair (s_0, v_2) with $v_2(x) = 0.7$, we have that, for all $t \in \mathbb{R}^+$, $(a, t) \in \mathbf{Enabled}_{\mathcal{ST}}^{timed}(s_0, v_2)$ and $(b, t) \in \mathbf{Enabled}_{\mathcal{ST}}^{timed}(s_0, v_2)$, thanks to the arcs $(s_0, (b, \top, \emptyset), s_0)$ and $(s_0, (a, \top, \{x\}), s_1)$ and the fact that we always have $v + t \models \top$.

Thanks to those definitions, we can lift the definition of $T\Sigma_f^\star$ (see Definition 2.126) for a strategy given as a deterministic STS whose locations are all accepting.

Definition 2.144. *For a deterministic STS $\mathcal{ST} = (S, s_0, \Delta)$, we let $T\Sigma_{\mathcal{ST}}^\star$ be the set of timed words consistent with $\mathcal{ST}$. It is defined as the smallest set of timed words containing the empty timed word, and closed by the following operations. For all $\theta \in T\Sigma_{\mathcal{ST}}^\star$:*

- *if* $\Delta(s_0, \theta) = (s, v)$ *is defined, and* $\boldsymbol{Enabled}^{timed}_{\mathcal{ST}}(s, v) \cap (\Sigma_C \times \mathbb{R}^+) \neq \emptyset$*:*
  for all $(a, t) \in \boldsymbol{Enabled}^{timed}_{\mathcal{ST}}(s, v) \cap (\Sigma_C \times \mathbb{R}^+)$*, we let* $\theta \cdot (a, T + t) \in T\Sigma^\star_{\mathcal{ST}}$ *and* $\theta \cdot (b, T + t') \in T\Sigma^\star_{\mathcal{ST}}$ *for all* $t' \leqslant t$ *and* $b \in \Sigma_E$*;*

- *if* $\Delta(s_0, \theta) = (s, v)$ *is defined, and* $\boldsymbol{Enabled}^{timed}_{\mathcal{ST}}(s, v) \cap (\Sigma_C \times \mathbb{R}^+) = \emptyset$ *(i.e. the controller does not want to play at this step):*
  we let $\theta \cdot (b, T + t') \in T\Sigma^\star_{\mathcal{ST}}$ *for all* $(b, T + t') \in \Sigma_E \times \mathbb{R}^+$*;*

- *if* $\Delta(s_0, \theta)$ *is not defined (i.e. the controller lost track of a move of the environment during the past):*
  we let $\theta \cdot \theta' \in T\Sigma^\star_{\mathcal{ST}}$ *for all* $\theta' \in T\Sigma^\star$ *(i.e. we declare that every possible future is valid);*

where $T = 0$ *if* $\theta = \varepsilon$*, and* $(c, T) \in \Sigma \times \mathbb{R}^+$ *is the last letter of* θ *otherwise.*

We are now able to define the *implementable reactive synthesis problem.*

Definition 2.145. *The MTL* implementable reactive synthesis problem *(IRS) is to decide, given an MTL formula* Φ*, whether there exists a set of clocks* X*, a symbolic alphabet* Γ *based on* (Σ, X)*, and a deterministic STS* $\mathcal{ST} = (S, s_0, \Delta)$ *over* Γ *whose locations are all accepting and such that every non-empty timed word consistent with* $\mathcal{ST}$ *satisfies formula* Φ*, i.e.:*

$$T\Sigma^\star_{\mathcal{ST}} \subseteq [\![\Phi]\!] \cup \{\varepsilon\}.$$

Example 2.146. Let us consider again the partitioned alphabet $\Sigma = \Sigma_C \cup \Sigma_E$, with $\Sigma_C = \{b\}$ and $\Sigma_E = \{a\}$, and the MTL formula:

$$\Phi \;\equiv\; \Box\big((a \wedge \Diamond_{>0} a \wedge \Box_{]0,1]} \neg a) \;\Rightarrow\; (\Diamond_{=1} b)\big).$$

This formula is realizable for IRS thanks to the set of clocks $X = \{x\}$, the symbolic alphabet Γ based of (Σ, X) and the STS $\mathcal{ST}_\Phi$ of Figure 2.27. Indeed, on the one hand, for a valuation v over X such that $v(x) = t \leqslant 1$:

$$\mathbf{Enabled}^{timed}_{\mathcal{ST}_\Phi}(s_0, v) \cap (\Sigma_C \times \mathbb{R}^+) = \{(b, 1 - t)\}.$$

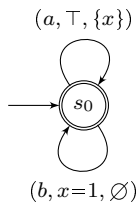


Figure 2.27: The STS $\mathcal{ST}_\Phi$.

So, a timed word $\theta \in T\Sigma^\star_{\mathcal{ST}_\Phi}$ such that $\Delta(s_0, \theta) = (s_0, v)$ can only be extended by $(b, T + 1 - t)$ and by all $(a, T + t')$ for $0 \leqslant t' \leqslant 1 - t$, where $T = 0$ if $\theta = \varepsilon$ and $(c, T) \in \Sigma \times \mathbb{R}^+$ is the last letter of θ otherwise. On the other hand, for a valuation v' over X such that $v(x) = t > 1$:

$$\mathbf{Enabled}^{timed}_{\mathcal{ST}_\Phi}(s_0, v) \cap (\Sigma_C \times \mathbb{R}^+) = \varnothing.$$

So, a timed word $\theta \in T\Sigma^\star_{\mathcal{ST}_\Phi}$ such that $\Delta(s_0, \theta) = (s_0, v')$ can be extended by all $(a, T + t')$ for $t' > 0$, where $T = 0$ if $\theta = \varepsilon$ and $(c, T) \in \Sigma \times \mathbb{R}^+$ is the last letter of θ otherwise.

Here are three examples of words of $T\Sigma^\star_{\mathcal{ST}_\Phi}$. They represent possible plays (according to the STS $\mathcal{ST}_\Phi$ of Figure 2.27) between the controller and the environment:

$$(a, 0)\,(a, 0.2)\,(a, 1.1)$$
$$(a, 0)\,(a, 0.2)\,(b, 1.2)$$
$$(a, 0.3)\,(b, 1.3)\,(a, 9)\,(b, 10)\,(a, 10.1).$$

In fact, it is easy to see that:

$$\begin{aligned} T\Sigma^\star_{\mathcal{ST}_\Phi} = \; & \{\, \theta = (\overline{\sigma}, \overline{\tau}) \mid \\ & \Big(\forall 1 \leqslant i \leqslant |\theta| : (\sigma_i = a) \;\Rightarrow\; (\,(i = |\theta|) \;\vee\; (\sigma_{i+1} = b \wedge \tau_{i+1} = \tau_i + 1)\,)\Big) \\ & \text{or } \Big(\exists 1 \leqslant i \leqslant |\theta| : (\sigma_i = a \wedge \sigma_{i+1} = a \wedge \tau_{i+1} \leqslant \tau_i + 1)\Big) \,\}\,. \end{aligned}$$

Hence, $T\Sigma^\star_{\mathcal{ST}_\Phi} \subseteq [\![\Phi]\!] \cup \{\varepsilon\}$.

A further refined problem consists in allowing the controller to use only a finite amount of *resources*, represented by the symbolic alphabet Γ. Indeed, we

may want to restrict the set X of clocks that the controller is allowed to use, as well as the set of constants c appearing in the clock constraints of the STS $\mathcal{ST}$. To this extent, as proposed by [17, 31], we first define the notion of *granularity*.

Definition 2.147. *A* granularity *is a triple $\mu = (X, m, K)$ where X is a finite set of clocks, $m \in \mathbb{N}\backslash\{0\}$, and $K \in \mathbb{N}$.*
A constraint g is μ-granular if $g \in \mathcal{G}(X)$ and each constant c occurring in g is of the form $\frac{\alpha}{m}$ with an integer $\alpha \leqslant K$. By extension, a symbolic alphabet Γ is μ-granular if it is based on (Σ, X) for an alphabet Σ, and the constraint of each symbolic letter in Γ is μ-regular.

Definition 2.148. *The MTL* bounded reactive synthesis problem *(BRS) is to decide, given an MTL formula Φ and a granularity $\mu = (X, m, K)$, whether there exists a μ-granular symbolic alphabet Γ based on (Σ, X), and a deterministic STS $\mathcal{ST}$ over Γ such that every non-empty timed word consistent with $\mathcal{ST}$ satisfies formula Φ, i.e.:*

$$T\Sigma^{\star}_{\mathcal{ST}} \subseteq [\![\Phi]\!] \cup \{\varepsilon\}.$$

Example 2.149. Let us consider the granularity $\mu = (\{x\}, 1, 1)$. We observe again the partitioned alphabet $\Sigma = \Sigma_C \cup \Sigma_E$, with $\Sigma_C = \{b\}$ and $\Sigma_E = \{a\}$, and the MTL formula:

$$\Phi \;\equiv\; \Box\big((a \wedge \Diamond_{>0} a \wedge \Box_{]0,1]} \neg a) \;\Rightarrow\; (\Diamond_{=1} b)\big).$$

This formula is realizable for BRS with μ, thanks to the μ-granular symbolic alphabet $\Gamma = \{(a, \top, \{x\}), (b, \top, \varnothing), (b, x = 1, \varnothing), (a, \top, \varnothing)\}$ and the deterministic STS $\mathcal{ST}_\Phi$ over Γ (see Figure 2.27 and Example 2.146 for details).
Let us now consider $\mu' = (\varnothing, 1, 1)$. Intuitively, Φ is *not* realizable for BRS with μ' because the controller is not able to remember the elapsing of exactly 1 time unit after the environment played an a.

Reactive synthesis problems with plants. D'Souza and Madhusudan [31], as well as Bouyer, Bozzelli and Chevalier [17], proposed an extension of the IRS and BRS problems defined above. Their aim was to restrain the evolution of runs

by a deterministic timed automaton, generally called a *plant* in this context. This plant will represent all the possible sequences of events over $\Sigma = \Sigma_C \cup \Sigma_E$ that may happen. Hence, it will limit, at each instant, the possible actions of the controller as well as those of the environment. Another interest of the plant is that, thanks to its non-accepting locations, it enables to limit the finite words on which the specification must be satisfied, which may relax constrains for the controller.

A notable difference with our previous definitions is that, in [31] and [17], the environment is allowed to play an enabled timed action *even if* its delay is higher than that in which the controller would like to play. We found this part of the definitions of [31] and [17] rather counter intuitive. Indeed, using those definitions and supposing that the controller would like to play a controllable action a in one time unit, the environment might 'pre-empt' the controller by playing an uncontrollable action b in *two* time units.

In the sequel, we adapt the definitions of [31] and [17] in way the environment is only allowed to play a timed action if its delay is smaller than that in which the controller would like to play. Other authors used this kind of concurrency between the controller and the environment (for instance, [28]). Such a definition seems more convenient to enable to simply and intuitively model plants in way to perform reactive synthesis with plant.

To be complete regarding the previous definitions of RS, IRS and BRS, we start by defining a general problem of *reactive synthesis with plant* (RSPlant) before to define the *implementable reactive synthesis with plant* (IRSPlant) and finally the *bounded reactive synthesis with plant* (BRSPlant). However, only the IRSPlant and BRSPlant were investigated in [31] and [17].

We start by formally defining a plant, which is a deterministic STS with finitely many locations and without deadlock. As we have already seen, such an STS corresponds exactly to a timed automaton.

Definition 2.150. *A* plant *is a deterministic STS* $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$*, over a*

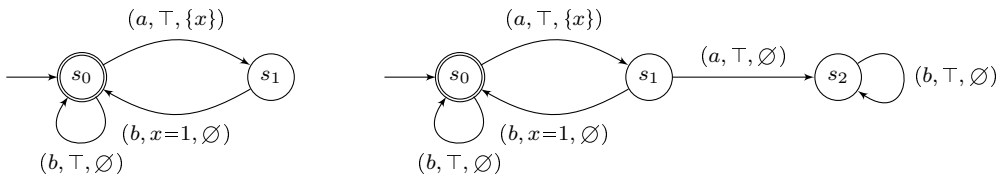


Figure 2.28: The STS $\mathcal{ST}$ (left) and the plant $\mathcal{P}$ (right).

symbolic alphabet Γ *based on* (Σ, X)*, such that, for each state* $p \in P$ *and each valuation* v *over* X*, there exists a timed action* $(a, t) \in \Sigma \times \mathbb{R}^+$ *and a transition* $(p, (a, g, R), p') \in \delta$ *such that* $v + t \models g$*.*

Example 2.151. The STS $\mathcal{ST}$ of Figure 2.28 (left) is not a plant. Indeed, when considering its location s_1 and the valuation v over $\{x\}$ such that $v(x) = 1.2$ there exists no timed action enabling to take the unique arc $(s_1, (b, x = 1, \varnothing), s_0)$ starting from s_1.
However, the STS $\mathcal{P}$ of Figure 2.28 (right) is indeed a plant.

As we did for RS, we can define the set of finite timed words consistent with the strategy f of the controller (see Definition 2.125) and the plant $\mathcal{P}$.

Definition 2.152. *The set of finite timed words* consistent *with the strategy* f *of the controller is the smallest set of timed words* $T\Sigma^\star_{f,\mathcal{P}}$ *containing the empty timed word* ε*, and closed by the following operations. For all* $\theta \in T\Sigma^\star_{f,\mathcal{P}}$*, letting* $(p, v_\mathcal{P}) = \delta(p_0, \theta)$ *(see Remark 2.141 ; it will always be defined, by induction):*

- *if* $f(\theta) = (a, t) \in \boldsymbol{Enabled}^{timed}_\mathcal{P}(p, v_\mathcal{P})$*:*
  we let $\theta \cdot (a, T + t) \in T\Sigma^\star_{f,\mathcal{P}}$ *and* $\theta \cdot (b, T + t') \in T\Sigma^\star_{f,\mathcal{P}}$ *for all* $t' \leqslant t$ *and* $b \in \Sigma_E$ *such that* $(b, t') \in \boldsymbol{Enabled}^{timed}_\mathcal{P}(p, v_\mathcal{P})$*;*

- *if* $f(\theta) = \bot$ *and* $\boldsymbol{Enabled}^{timed}_\mathcal{P}(p, v_\mathcal{P}) \cap (\Sigma_E \times \mathbb{R}^+) \neq \varnothing$ *(i.e. the controller does not want to play, but the environment has some enabled actions):*
  we let $\theta \cdot (b, T + t') \in T\Sigma^\star_{f,\mathcal{P}}$ *for all* $(b, t') \in \boldsymbol{Enabled}^{timed}_\mathcal{P}(p, v_\mathcal{P}) \cap (\Sigma_E \times \mathbb{R}^+)$*;*

- *otherwise, (either the controller proposed a non-enabled actions, or it proposed $\perp$ whereas the environment had no enabled actions):*
  we let $\theta \cdot \theta' \in T\Sigma^\star_{f,\mathcal{P}}$ for all $\theta \cdot \theta' \in L(\mathcal{P})$ (i.e. we declare that every possible future of the plant is valid);

where $T = 0$ if $\theta = \varepsilon$, and $(c, T) \in \Sigma \times \mathbb{R}^+$ is the last letter of θ otherwise.

We are now able to define the *reactive synthesis problem with plant.*

Definition 2.153. *The MTL* reactive synthesis problem with plant *(RSPlant) is to decide, given an MTL formula Φ and a plant $\mathcal{P}$, whether there exists a strategy f of the controller such that every non-empty timed word that is (i) consistent with strategy f and the plant $\mathcal{P}$ and (ii) accepted by the plant*[3]*, satisfies the formula Φ, i.e.:*

$$T\Sigma^\star_{f,\mathcal{P}} \cap L(\mathcal{P}) \subseteq [\![\Phi]\!] \cup \{\varepsilon\}.$$

Example 2.154. Let us consider again the partitioned alphabet $\Sigma = \Sigma_C \cup \Sigma_E$, where $\Sigma_C = \{b\}$ and $\Sigma_E = \{a\}$, and the MTL formula:

$$\Phi \;\equiv\; \Box\big((a \wedge \Diamond_{>0} a \wedge \Box_{]0,1]} \neg a) \;\Rightarrow\; (\Diamond_{=1} b)\big).$$

We furthermore consider the plant $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$ of Figure 2.29. We define the following strategy f of the controller: for all $\theta \in T\Sigma^\star$, $f(\theta) = (b, 1)$. Remark that $(b, 1)$ is in $\mathbf{Enabled}^{timed}_{\mathcal{P}}(p_0, v_{\mathcal{P}})$ and in $\mathbf{Enabled}^{timed}_{\mathcal{P}}(p_1, v_{\mathcal{P}})$ for all $\theta \in T\Sigma^\star$ such that $\delta(p_0, \theta)$ is equal to $(p_0, v_{\mathcal{P}})$ or $(p_1, v_{\mathcal{P}})$. So, in both cases, $\theta \cdot (b, T+1)$ is added to $T\Sigma^\star_{f,\mathcal{P}}$, as well as $\theta \cdot (a, T + t')$ for all $t' \leqslant 1$, where $T = 0$ if $\theta = \epsilon$ and $(c, T) \in \Sigma \times \mathbb{R}^+$ is the last letter of θ otherwise. Such words of the form $\theta \cdot (a, T + t')$ prevent the left-hand side of the implication of Φ to be verified. So, all their extensions satisfy Φ (it is important because starting from location p_2, $(b, T + 1)$ is not an enabled action for the controller and the definition of $T\Sigma^\star_{f,\mathcal{P}}$ hence allows every possible future of the plant). Hence, $T\Sigma^\star_{f,\mathcal{P}} \cap L(\mathcal{P}) \subseteq [\![\Phi]\!] \cup \{\varepsilon\}$

[3] not all the states of $\mathcal{P}$ are accepting.

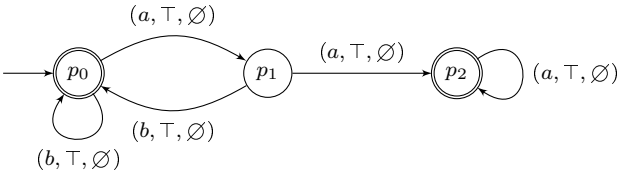


Figure 2.29: A plant $\mathcal{P}$.

and Φ is realizable for RSPlant.
The MTL formula

$$\Phi' \equiv \Box\big((a \wedge \Diamond_{>1} a) \Rightarrow (\Diamond_{=1} b)\big),$$

is also realizable for RSPlant with $\mathcal{P}$, using the strategy f presented above. This contrasts with the result of Example 2.128 in which a more complex strategy was required.

In the following, in way to define the IRSPlant and BRSPlant problems, we rewrite the definitions of [31] and [17], adding however a new kind of concurrency between the controller and the environment. Indeed, in our definitions, the environment is only allowed to play an enabled timed action if its delay is smaller than that in which the controller would like to play. This concurrency aims to avoid the counter intuitive situations in which the controller would like to play a controllable action a in one time unit, but the environment 'pre-empts' the controller by playing an uncontrollable action b in *two* time units (which was allowed by the definitions of [31] and [17]).

From RSPlant, we define IRSPlant considering that controllers are described by STS (as in the definition of IRS: Definition 2.145) whose locations are all accepting. IRSPlant is simply defined adapting the definition of *consistent timed words*. One can remark, reading the following definition, that a consistent time word is a word present in the parallel composition of the plant and the STS representation of the controller. This parallel composition is formally defined in [31] and [17], but avoided here to simplify the definition.

Definition 2.155. *For a deterministic STS $\mathcal{ST} = (S, s_0, \Delta)$ and a plant $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$, we let $T\Sigma^{\star}_{\mathcal{ST},\mathcal{P}}$ be the set of timed words consistent with $\mathcal{ST}$ and $\mathcal{P}$. It is defined as the smallest set of timed words containing the empty timed word, and closed by the following operations. For all $\theta \in T\Sigma^{\star}_{\mathcal{ST},\mathcal{P}}$, letting $(p, v_{\mathcal{P}}) = \delta(p_0, \theta)$ (see Remark 2.141 ; it will always be defined, by induction),*

- *if $\Delta(s_0, \theta) = (s, v)$ is defined and $\boldsymbol{Enabled}^{timed}_{\mathcal{ST}}(s, v) \cap \boldsymbol{Enabled}^{timed}_{\mathcal{P}}(p, v_{\mathcal{P}}) \cap (\Sigma_C \times \mathbb{R}^+) \neq \emptyset$:*
  for all $(a, t) \in \boldsymbol{Enabled}^{timed}_{\mathcal{ST}}(s, v) \cap \boldsymbol{Enabled}^{timed}_{\mathcal{P}}(p, v_{\mathcal{P}}) \cap (\Sigma_C \times \mathbb{R}^+)$, we let $\theta \cdot (a, T + t) \in T\Sigma^{\star}_{f,\mathcal{P}}$ and $\theta \cdot (b, T + t') \in T\Sigma^{\star}_{f,\mathcal{P}}$ for all $t' \leqslant t$ and $b \in \Sigma_E$ such that $(b, t') \in \boldsymbol{Enabled}^{timed}_{\mathcal{P}}(p, v_{\mathcal{P}})$;

- *if $\Delta(s_0, \theta) = (s, v)$ is defined and $\boldsymbol{Enabled}^{timed}_{\mathcal{ST}}(s, v) \cap \boldsymbol{Enabled}^{timed}_{\mathcal{P}}(p, v_{\mathcal{P}}) \cap (\Sigma_C \times \mathbb{R}^+) = \emptyset$ and $\boldsymbol{Enabled}^{timed}_{\mathcal{P}}(p, v_{\mathcal{P}}) \cap (\Sigma_E \times \mathbb{R}^+) \neq \emptyset$ (i.e. the controller does not want to play, but the environment has some enabled actions):*
  we let $\theta \cdot (b, T + t') \in T\Sigma^{\star}_{\mathcal{ST},\mathcal{P}}$ for all $(b, t') \in \boldsymbol{Enabled}^{timed}_{\mathcal{P}}(p, v_{\mathcal{P}}) \cap (\Sigma_E \times \mathbb{R}^+)$;

- *otherwise (i.e. either the controller proposed only non-enabled actions while the environment had no enabled actions, or the controller lost track of a move of the environment during the past):*
  we let $\theta \cdot \theta' \in T\Sigma^{\star}_{\mathcal{ST},\mathcal{P}}$ for all $\theta \cdot \theta' \in L(\mathcal{P})$ (i.e. we declare that every possible future of the plant is valid);

where $T = 0$ if $\theta = \varepsilon$, and $(c, T) \in \Sigma \times \mathbb{R}^+$ is the last letter of θ otherwise.

We are now ready to define the implementable reactive synthesis problem with plant.

Definition 2.156. *The MTL* implementable reactive synthesis problem with plant *(IRSPlant) is to decide, given an MTL formula Φ and a plant $\mathcal{P}$ whose set of clocks is $X_{\mathcal{P}}$, whether there exists a set of clocks X_C, a symbolic alphabet*

Γ based on $(\Sigma, X_{\mathcal{P}} \cup X_C)$, and a deterministic STS $\mathcal{ST}$ over Γ such that every non-empty timed word consistent with $\mathcal{ST}$ and $\mathcal{P}$ satisfies formula Φ, i.e.:

$$T\Sigma^{\star}_{\mathcal{ST},\mathcal{P}} \cap L(\mathcal{P}) \subseteq [\![\Phi]\!] \cup \{\varepsilon\}.$$

Example 2.157. Once again, we consider the partitioned alphabet $\Sigma = \Sigma_C \cup \Sigma_E$, where $\Sigma_C = \{b\}$ and $\Sigma_E = \{a\}$, the MTL formula

$$\Phi \;\equiv\; \Box\big((a \wedge \Diamond_{>0} a \wedge \Box_{]0,1]} \neg a) \;\Rightarrow\; (\Diamond_{=1} b)\big)$$

and the plant $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$ of Figure 2.30 (left). Φ is realizable for IRSPlant thanks to the empty set of clocks $X = \varnothing$, the symbolic alphabet Γ based of $(\Sigma, \{x\})$ and the STS $\mathcal{ST}_\Phi = (S, s_0, \Delta)$ of Figure 2.31 (left).
Indeed, on the one hand, for a valuation v over X such that $v(x) = t \leqslant 1$:

$$\begin{aligned} & \mathbf{Enabled}^{timed}_{\mathcal{ST}_\Phi}(s_0, v) \cap \mathbf{Enabled}^{timed}_{\mathcal{P}}(p_0, v_{\mathcal{P}}) \cap (\Sigma_C \times \mathbb{R}^+) \\ = \; & \mathbf{Enabled}^{timed}_{\mathcal{ST}_\Phi}(s_0, v) \cap \mathbf{Enabled}^{timed}_{\mathcal{P}}(p_1, v_{\mathcal{P}}) \cap (\Sigma_C \times \mathbb{R}^+) \\ = \; & \{(b, 1-t)\}. \end{aligned}$$

So, a timed word $\theta \in T\Sigma^{\star}_{\mathcal{ST}_\Phi}$ such that $\delta(s_0, \theta) = (p_0, v)$ can be extended by all $(a, T+t)$ and by $(b, T+1-t)$, while a timed word $\theta \in T\Sigma^{\star}_{\mathcal{ST}_\Phi}$ such that $\delta(s_0, \theta) = (p_1, v)$ can only be extended by $(b, T+1-t)$, where $T = 0$ if $\theta = \varepsilon$ and $(c, T) \in \Sigma \times \mathbb{R}^+$ is the last letter of θ otherwise.
On the other hand, for a valuation v' over X such that $v(x) = t > 1$:

$$\begin{aligned} & \mathbf{Enabled}^{timed}_{\mathcal{ST}_\Phi}(s_0, v) \cap \mathbf{Enabled}^{timed}_{\mathcal{P}}(p_0, v_{\mathcal{P}}) \cap (\Sigma_C \times \mathbb{R}^+) \\ = \; & \mathbf{Enabled}^{timed}_{\mathcal{ST}_\Phi}(s_0, v) \cap \mathbf{Enabled}^{timed}_{\mathcal{P}}(p_1, v_{\mathcal{P}}) \cap (\Sigma_C \times \mathbb{R}^+) \\ = \; & \varnothing. \end{aligned}$$

So, a timed word $\theta \in T\Sigma^{\star}_{\mathcal{ST}_\Phi}$ such that $\delta(s_0, \theta) = (p_0, v)$ can be extended by all $(a, T+t)$, where $T = 0$ if $\theta = \varepsilon$ and $(c, T) \in \Sigma \times \mathbb{R}^+$ is the last letter of θ otherwise.[4]
It is so easy to see that

$$\begin{aligned} T\Sigma^{\star}_{\mathcal{ST}_\Phi,\mathcal{P}} \;=\; \{ \;\; & \theta = (\overline{\sigma}, \overline{\tau}) \;\mid\; \forall 1 \leqslant i \leqslant |\theta| : \\ & (\sigma_i = a) \;\Rightarrow\; \big((i = |\theta|) \vee (\sigma_{i+1} = b \wedge \tau_{i+1} = \tau_i + 1)\big) \;\; \} \end{aligned}$$

[4]Remark that the case of a timed word $\theta \in T\Sigma^{\star}_{\mathcal{ST}_\Phi}$ such that $\delta(s_0, \theta) = (p_1, v)$ never happens by what precedes.

Figure 2.30: A plant $\mathcal{P}$ (left) and a plant $\mathcal{P}'$ (right).

Figure 2.31: The deterministic STS $\mathcal{ST}_\Phi$ (left) and $\mathcal{ST}'_\Phi$ (right).

So that $T\Sigma^\star_{\mathcal{ST}_\Phi,\mathcal{P}} \cap L(\mathcal{P}) \subseteq [\![\Phi]\!] \cup \{\varepsilon\}$.
Let us notice that Φ is realizable for this IRSPlant problem thanks to the fact that, even if the controller has no proper clock, it is allowed to observe the values taken by the clock x of the environment at any time.
Considering the IRSPlant problem from formula Φ and the plant $\mathcal{P}'$ of Figure 2.30 (right), Φ is also realizable for IRSPlant. Neveretheless, it is now necessary to use the set of clocks $X = \{y\}$, the symbolic alphabet Γ based of $(\Sigma, \{y\})$ and the STS $\mathcal{ST}'_\Phi$ of Figure 2.31 (right).

Finally, the *bounded reactive synthesis problem with plant* (BRSPlant) is obtained from IRSPlant, by *a priori* fixing a granularity μ, representing the finite amount of *resources* the controller is only allowed to use. This adaptation is similar to that from the definiton of IRS to that of BRS (see Definitions 2.145 and 2.148).

Definition 2.158. *The MTL* bounded reactive synthesis problem *(BRSPlant) is to decide, given an MTL formula Φ, a plant $\mathcal{P}$ of set of clocks $X_\mathcal{P}$ and a granularity $\mu = (X_\mathcal{P} \cup X_C, m, K)$, whether there exists a μ-granular symbolic*

alphabet Γ *based on* (Σ, X)*, and a deterministic STS* $\mathcal{ST}$ *over* Γ *such that every non-empty timed word consistent with* $\mathcal{ST}$ *and* $\mathcal{P}$ *satisfies formula* Φ*, i.e.:*

$$T\Sigma^{\star}_{\mathcal{ST},\mathcal{P}} \cap L(\mathcal{P}) \subseteq [\![\Phi]\!] \cup \{\varepsilon\}.$$

Example 2.159. We observe again the partitioned alphabet $\Sigma = \Sigma_C \cup \Sigma_E$, with $\Sigma_C = \{b\}$ and $\Sigma_E = \{a\}$, the MTL formula

$$\Phi \;\equiv\; \Box\big((a \wedge \Diamond_{>0} a \wedge \Box_{]0,1]} \neg a) \;\Rightarrow\; (\Diamond_{=1} b)\big)$$

and the plant $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$ of Figure 2.30 (left). Let us consider the granularity $\mu = (\{x\}, 1, 1)$, i.e. the controller has no proper clock. Φ is realizable for BRSPlant with μ, thanks to the μ-granular symbolic alphabet $\Gamma = \{(a, \top, \{x\}), (b, \top, \varnothing), (b, x = 1, \varnothing)\}$ and the deterministic STS $\mathcal{ST}_\Phi$ over Γ (see Figure 2.31 (left) and Example 2.157 for details).
Let us now consider the plant $\mathcal{P}'$ of Figure 2.30 (right) and the granularity $\mu = (\{y\}, 1, 1)$, i.e. the controller has one proper clock y. Φ is realizable for BRSPlant with μ', thanks to the μ'-granular symbolic alphabet $\Gamma = \{(a, \top, \{y\}), (b, \top, \varnothing), (b, y = 1, \varnothing)\}$ and the deterministic STS $\mathcal{ST}'_\Phi$ over Γ (see Figure 2.31 (right) and Example 2.157 for details).

Remark 2.160. When considering any real-time logic L, the L RS, IRS, BRS, RSPlant, IRSPlant and BRSPlant problems are defined similarly as the MTL corresponding problems, for a formula Φ of L.

Here are some decidability results about the MTL IRSPlant and BRSPlant problems: they are the same as those presented in [17], even if the definitions of these problems were slightly different from ours. Indeed, our definitions are simplified because we avoided the construction of the actual parallel composition between the plant and the controller: if the presentation is different, from this point of view, our definitions are equivalent to that of [17]. The unique real difference is the new kind of *concurrency* between the controller and the environment: we only allow the environment to play an action with a delay smaller than that of the strategy of the controller, while the environment is allowed to play with

any time delay in [17]. It is not difficult to see that this slight modification of the definitions of [17] do not affect the following decidabillity results[5].

Theorem 2.161. *The MTL IRSPlant problem is undecidable.*

Theorem 2.162. *The MTL BRSPlant problem is decidable.*

2.9 From MTL to OCATA

In this section, we come back to the unique decidable version of the MTL model-checking problem: on the pointwise semantics, over finite words. Let us consider a system represented by a timed automaton $\mathcal{B}$ and a property given by an MTL formula Φ. We want to verify if $L(\mathcal{B}) \cap [\![\neg\Phi]\!] = \emptyset$. Ouaknine and Worrell presented in [51] a non primitive recursive algorithm to decide this version of the MTL model-checking problem. Their technique is based on the construction, from $\neg\Phi$, of an OCATA $\mathcal{A}_{\neg\Phi}$ recognizing $[\![\neg\Phi]\!]$. They then define a timed transition system representing the parallel execution of $\mathcal{B}$ and $\mathcal{A}_{\neg\Phi}$. Their MTL model-checking algorithm then relies on a region construction and a well quasi order enabling to stop the branches of this (infinite) timed transition system.

In the present section, we explain the method of Ouaknine and Worrell ([51]) to build, from any MTL formula Φ (in negative normal form), an OCATA $\mathcal{A}_\Phi$ whose language on *finite timed words* is exactly the finite word language of Φ, i.e.: $L(\mathcal{A}_\Phi) = [\![\Phi]\!]$.

To formally define $\mathcal{A}_\Phi$, observe that we can transform any MTL formula in an equivalent MTL formula in *negative normal form* (in which negation can only be present on letters $\sigma \in \Sigma$) using the operators: $\wedge, \vee, \neg, U_I$ and $\tilde{U}_I$.

[5] only the function *valid* defined in [17], and the notion of *valid set of successors* of [31], must be adapted, which does not affect the proofs of these papers.

Example 2.163. For instance, the MTL formula $\Phi \equiv \Box(p \Rightarrow \Diamond_{[2,3]} q)$ is equivalent to the formula $\bot \tilde{U}_{[0,+\infty[}(\neg p \vee \top U_{[2,3]} q)$, which is in negative normal form. Its negation, $\neg\big(\Box(p \Rightarrow \Diamond_{[2,3]} q)\big)$, is equivalent to the following negative normal form MTL formula: $\top U_{[0,+\infty[}(p \wedge \bot \tilde{U}_{[2,3]} \neg q)$.

Definition 2.164. *For an MTL formula Φ in negative normal form, we let $\mathcal{A}_\Phi = (\Sigma, L, \ell_0, F, \delta)$ where: L is the set containing the initial copy of Φ, noted 'Φ_{init}', and all the formulas of $Sub(\Phi)$ whose outermost connective is 'U' or '$\tilde{U}$'; $\ell_0 = \Phi_{init}$; F is the set of the elements of L of the form $\Phi_1 \tilde{U}_I \Phi_2$. Finally δ is defined*[6] *by induction on the structure of Φ:*

- $\delta(\Phi_{init}, \sigma) = x.\delta(\Phi, \sigma)$
- $\delta(\Phi_1 \vee \Phi_2, \sigma) = \delta(\Phi_1, \sigma) \vee \delta(\Phi_2, \sigma)$
- $\delta(\Phi_1 \wedge \Phi_2, \sigma) = \delta(\Phi_1, \sigma) \wedge \delta(\Phi_2, \sigma)$
- $\delta(\Phi_1 U_I \Phi_2, \sigma) = (x.\delta(\Phi_2, \sigma) \wedge x \in I) \vee (x.\delta(\Phi_1, \sigma) \wedge \Phi_1 U_I \Phi_2 \wedge x \leqslant sup(I))$
- $\delta(\Phi_1 \tilde{U}_I \Phi_2, \sigma) = (x.\delta(\Phi_2, \sigma) \vee x \notin I) \wedge (x.\delta(\Phi_1, \sigma) \vee \Phi_1 \tilde{U}_I \Phi_2 \vee x > sup(I))$
- $\forall \sigma_1, \sigma_2 \in \Sigma$:
  $\delta(\sigma_1, \sigma_2) = \begin{cases} true & \text{if } \sigma_1 = \sigma_2 \\ false & \text{if } \sigma_1 \neq \sigma_2 \end{cases}$ *and* $\delta(\neg\sigma_1, \sigma_2) = \begin{cases} false & \text{if } \sigma_1 = \sigma_2 \\ true & \text{if } \sigma_1 \neq \sigma_2 \end{cases}$
- $\forall \sigma \in \Sigma$: $\delta(\top, \sigma) = \top$ *and* $\delta(\bot, \sigma) = \bot$.

Example 2.165. As an example, consider the formula $\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,2]} b)$, which is a shorthand for $\bot \tilde{U}_{[0,+\infty[}\big(a \Rightarrow (\top U_{[1,2]} b)\big)$. The OCATA $\mathcal{A}_\Phi$ is given in Figure 2.32, where the location $\ell_\Box$ corresponds to Φ and the location $\ell_\Diamond$ corresponds to $\top U_{[1,2]} b$. One can check that this automaton follows strictly the above definition, after simplification of the formulas. Observe the edge labelled by '$b, x \in [1,2]$' from $\ell_\Diamond$, without target location: it depicts the fact that

[6] Remark that the $x \leqslant sup(I)$ and $x > sup(I)$ conditions in the respective definitions of $\delta(\Phi_1 U_I \Phi_2, \sigma)$ and $\delta(\Phi_1 \tilde{U}_I \Phi_2, \sigma)$ have been added here for technical reasons. This does not modify the accepted language. Indeed, in [51], these conditions are given in the infinite word semantics of OCATA.

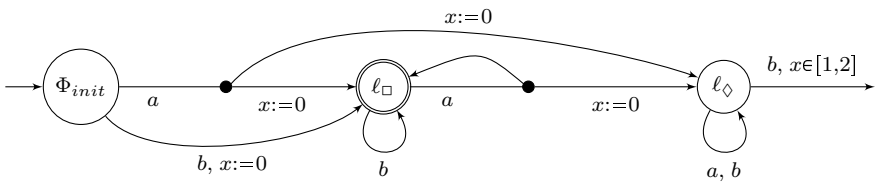


Figure 2.32: OCATA $\mathcal{A}_\Phi$ with $\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,2]} b)$.

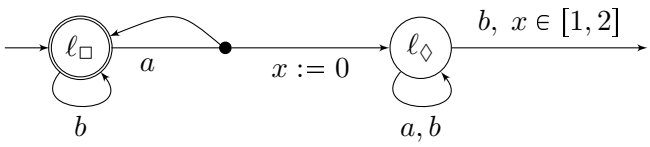


Figure 2.33: OCATA $\mathcal{A}$ with $L(\mathcal{A}) = [\![\, \Box(a \Rightarrow \Diamond_{[1,2]} b) \,]\!]$.

$\delta(\top U_{[1,2]} b, b) = (x \in [1,2]) \vee (\top U_{[1,2]} b)$. Intuitively, when the automaton has a copy in location '$\Diamond$' with a clock valuation in $[1,2]$, the copy can be *removed*, because a minimal model of $x \in [1,2]$ with respect to a valuation v with $v \in [1,2]$ is $\varnothing$.

The OCATA $\mathcal{A}_\Phi$ may be simplified, without modifying the accepted language (in the present case), removing the location 'Φ_{init}' and using the location '$\ell_\Box$' as initial location instead. We so obtain the OCATA $\mathcal{A}$ of Figure 2.33. This OCATA will be often used in the sequel to represent $[\![\Phi]\!]$, in way to enhance readability of the examples: it will be called $\mathcal{A}_\Phi$ by abuse of language.

With this definition, we have:

Theorem 2.166 ([51]). *For all MTL formula Φ interpreted over finite words: $L(\mathcal{A}_\Phi) = [\![\Phi]\!]$.*

In Section 5.1.2, we extend this result to the setting of infinite words, showing that $L^\omega(\mathcal{A}_\Phi) = [\![\Phi]\!]^\omega$.

2.10 The continuous semantics

The definitions of timed automata, alternating timed automata and MTL semantics presented in the previous subsections are in fact defined in the setting of the *pointwise semantics*. Indeed, the contributions presented in this thesis only concern this pointwise semantics. Nevertheless, there exists an alternative semantics for these objects, called the *continuous semantics*, which is also frequently used in the literature. While the pointwise semantics is useful to characterize the events happening in precise instants, the continuous semantics is opportune to represent the state changings of a system. For the sake of completeness, we here present the continuous semantics of automata and MTL. We furthermore exhibit classical decidability and complexity results about the problems linked to MTL, on the continuous semantics.

2.10.1 Timed state sequences

In the continuous semantics, instead of observing timed words, with their letters read on precise instants, we consider *timed state sequences* that associate a letter with each instant $t \in \mathbb{R}^+$ (or to each instant in $[0, t^\star]$, for a certain $t^\star \in \mathbb{R}^+$, in the case of finite timed state sequences). Here is the formal definition of *timed state sequence*, based on the definition of *interval sequence*.

Definition 2.167. *An* interval sequence $\overline{I} = I_1 I_2 I_3 \ldots$ *is a finite or infinite word over* $\mathcal{I}(\mathbb{R})$ *such that:*

- *initiality:* I_1 *is left-closed and* $\inf(I_1) = 0$,
- *adjacency: for all* $i \geqslant 1$, *the intervals* I_i *and* I_{i+1} *are adjacent*[7],
- *progress: if* $\overline{I}$ *is infinite, for all* $t \in \mathbb{R}^+$, *there is* $i \geqslant 1$ *such that* $t \in I_i$.

[7] We recall that, for $J_1, J_2 \in \mathbb{R}$, with $J_1 = \langle_1 a, b \rangle_1$, $J_2 = \langle_2 c, d \rangle_2$ and $J_1 < J_2$, we say that J_1 and J_2 are *adjacent* iff $b = c$ and $J_1 \cup J_2 = \langle_1 a, d \rangle_2$.

Definition 2.168. *A* timed state sequence *is a pair* $\Theta = (\overline{\sigma}, \overline{I})$ *where* $\overline{\sigma}$ *is a word over* Σ*,* $\overline{I}$ *is an interval sequence and* $|\overline{\sigma}| = |\overline{I}|$*.*
We also note Θ *as* $(\sigma_1, I_1)(\sigma_2, I_2)(\sigma_3, I_3)\ldots$*, and let* $|\Theta| = |\overline{\sigma}|$*.*
We denote by $TS\Sigma$ *the set of all finite timed state sequences and by* $TS\Sigma^\omega$ *the set of all infinite timed state sequences over* Σ*.*

Let $I \in \mathcal{I}(\mathbb{R})$ and $t \in \mathbb{R}^+$, we note $t - I$ for $\{t - i \in \mathbb{R} \mid i \in I \text{ and } t \geqslant i\}$.
For a (finite or infinite) timed state sequence $\Theta = (\overline{\sigma}, \overline{I})$, where $\overline{\sigma} = \sigma_1\sigma_2\sigma_3\ldots$ and $\overline{I} = I_1 I_2 I_3 \ldots$, and for $t \in I_i$ the *suffix* Θ^t of Θ at time t is the timed state sequence consisting in $\sigma_i\sigma_{i+1}\sigma_{i+2}\ldots$ and $I_i - t \quad I_{i+1} - t \quad I_{i+2} - t \ \ldots$.

Definition 2.169. *A* continuous timed language *is a (possibly infinite) set of timed state sequences.*

Example 2.170. Let us consider the alphabet $\Sigma = \{a, b\}$. Let us note $\overline{\sigma} = aaabab$ and $\overline{I} = [0, 0.2[\ [0.2, 1]\ ]1, 3.2]\ ]3.2, 3.35[\ [3.35, 4.1[\ [4.1, 6[$. $\Theta = (\overline{\sigma}, \overline{I})$ is an element of $T\Sigma^\star$: it is a finite timed state sequence over Σ. We can also denote Θ as

$$(a, [0, 0.2[)(a, [0.2, 1])(a,]1, 3.2])(b,]3.2, 3.35[)(a, [3.35, 4.1[)(b, [4.1, 6[).$$

The length of Θ is equal to 6: $|\Theta| = 6$. The suffix $\Theta^{2.7}$ of Θ at time $2.7 \in]1, 3.2]$ is

$$\Theta^{2.7} = (a, [0, 0.5])(b,]0.5, 0.65[)(a, [0.65, 1.4[)(b, [1.4, 3.3[).$$

Let us note $\overline{\sigma}' = ababab\ldots$ and $\overline{I}' = [0, 1[\ [1, 2[\ [2, 3[\ [3, 4[\ [4, 5[\ [5, 6[\ldots$. $\Theta' = (\overline{\sigma}', \overline{I}')$ is an element of $T\Sigma^\omega$: it is an infinite timed state sequence over Σ. Θ' can also be denoted

$$\Theta' = (a, [0, 1[)(b, [1, 2[)(a, [2, 3[)(b, [3, 4[)(a, [4, 5[)(b, [5, 6[)\ldots$$

and $|\Theta'| = +\infty$. The suffix Θ^3 of Θ at time $3 \in [3, 4[$ is

$$\Theta^3 = (b, [0, 1[)(a, [1, 2[)(b, [2, 3[)\ldots.$$

Let us note $L_1 = \{\Theta\}$: it is a continuous timed language (of finite timed state sequences) over Σ. The set L_2 of infinite timed state sequences over Σ whose first letter is a is also a continuous timed language (of infinite timed state sequences):

$$L_2 = \{\Theta = (\overline{\sigma}, \overline{I}) \mid \sigma_1 = a \ \wedge \ |\Theta| = +\infty\}.$$

2.10.2 Timed automata and alternating timed automata

While timed automata are acceptors of timed words, they are not able to accept timed state sequences. In [5], an alternative definition of timed automata, for the continuous semantics, is given in way they accept timed state sequences. Another object, called *timed signal transducer* ([44]) is also able to recognize timed state sequences. We do not present their formal definition here and encourage the interested reader to consult these papers for further details.
In a similar way, alternating timed automata recognize timed words but not timed state sequences. As far as we know, no definition of alternating timed automata for the continuous semantics has ever been investigated.

2.10.3 Metric Temporal Logic

We present here the *continuous semantics* of MTL and investigate the MTL model-checking problems in this setting.

MTL continuous semantics. As well as for the MTL pointwise semantics (of Definition 2.119), the definition of the continuous semantics of MTL is based on the syntax of MTL presented in Definition 2.116. This semantics is adapted for the setting of finite words as well as for that of infinite words.

Definition 2.171 (Continuous semantics of MTL)**.** *Given a timed state sequence* $\Theta = (\overline{\sigma}, \overline{I})$ *over* Σ *and an MTL formula* Φ*, we say that* Θ *satisfies* Φ*, written* $\Theta \models \Phi$*, iff the following holds:*

1. $\Theta \models \top$,
2. $\Theta \models \sigma$ *iff* $\sigma_1 = \sigma$,
3. $\Theta \models \Phi_1 \wedge \Phi_2$ *iff* $\Theta \models \Phi_1$ *and* $\Theta \models \Phi_2$,
4. $\Theta \models \neg\Phi$ *iff* $\Theta \not\models \Phi$,
5. $\Theta \models \Phi_1 U_J \Phi_2$ *iff* $\exists t \in J$ *and* $I_i \in \overline{I}$ *such that* $t \in I_i$, $\Theta^t \models \Phi_2$ *and* $\forall t' \in [0, t[, \Theta^{t'} \models \Phi_1$.

We note $[\![\Phi]\!]_{cont} = \{\theta \mid |\theta| < \infty$ *and* $\theta \models \Phi\}$ *the* finite continuous timed language *of* Φ *and* $[\![\Phi]\!]^{\omega}_{cont} = \{\theta \mid |\theta| = +\infty$ *and* $\theta \models \Phi\}$ *the* infinite continuous timed language *of* Φ.

As for the pointwise semantics, the previously given semantics for the U operator is known as the *unstrict* continuous semantics. Here is the *strict* continuous semantics for the U operator:

6. $\Theta \models \Phi_1 U_J \Phi_2$ iff $\exists t \in J$ and I_i in the sequence $\overline{I}$ such that $t \in I_i$, $\Theta^t \models \Phi_2$ and $\forall t' \in]0, t[, \Theta^{t'} \models \Phi_1$.

In the sequel, when we do not specify the semantics used for the U, it means we consider the unstrict one.

While the pointwise and continuous semantics seem to be very close to each other, there exists subtle differences such that a same formula may have different meanings when interpreted over the pointwise semantics or the continuous one.

Example 2.172. The formula $\Phi \equiv \Box((\Diamond_{[1,1]} a) \Rightarrow b)$, interpreted over the continuous semantics, means that 'each a is preceded by a b one time unit earlier'. When interpreted over the pointwise semantics, its meaning changes. For example, the finite timed word '$(a, 3)$' satisfies Φ because no letter is read one time

unit before the a: hence, there is no position in the word that verifies $\Diamond_{[1,1]}a$. Interpreting the formula over the pointwise semantics, we so have no condition to verify in instant 2, contrary to the continuous semantics. So, in the pointwise semantics, the meaning of Φ would be that 'each a is preceded one time earlier, either by a b or by no letter'.

We recall that the definition of the continuous semantics of MTL is given here for the sake of completeness, but that all the contributions presented in this thesis only concern the pointwise semantics: their extension to the continuous setting is not clear. Indeed, the translation from an MTL formula Φ to an OCATA $\mathcal{A}_\Phi$ exhibited in Definition 2.164, which is starting point of our contributions, only works for MTL formulas interpreted *over the pointwise semantics*. As far as we know, no definition of alternating timed automata for the continuous semantics has ever been investigated.

The problems over MTL on the continuous semantics. We here consider the MTL satisfiability and model-checking problems on the continuous semantics. We first present the result from [5] stating that the MTL satisfiability is undecidable. We then discuss about the possible versions of the MTL model-checking problem.

Definition 2.173 ([5]). *Over the continuous semantics, the MTL satisfiability problem is undecidable.*

With the definition of the MTL continuous semantics, two new versions of the MTL model-checking should be investigated: the MTL model-checking problem with the continuous semantics over *finite* and *infinite* words. We here present the key ideas of proofs given in [16][8] to convince the reader that these problems are undecidable:

Theorem 2.174. *[7] The MTL model-checking problem over finite and infinite words with the continuous semantics are undecidable.*

[8] The original proof can be found in [7]

To simplify the key intuitions presented in the sequel, we will consider the strict (continuous) semantics of the U operator in this paragraph.[9].

Let us consider a perfect channel machine $\mathcal{C} = (S, s_0, M, C, \Delta)$, where S is a finite set of locations, $s_0 \in S$ is the initial location, M is a finite set of messages, C is a finite set of FIFO channels and $\Delta \subseteq S \times A \times S$ is the transition relation over the set of actions $A = \{c!m, c?m \mid c \in C \text{ and } m \in M\}$.
We first show how we can encode runs of a perfect channel machine by an MTL formula, following the ideas of [16]. Let us recall that the halting problem of a channel machine is undecidable over finite words. As it is straightforward, thanks to this encoding, to reduce this problem to the MTL model-checking problem over finite words, we will conclude that the MTL model-checking problem over finite words is undecidable.
A run of the machine is encoded by a timed state sequence, represented by an MTL formula, in the following way:

1. The first letter of the timed state sequence is the initial location of the machine:
$$\Phi_{init} = s_0.$$
2. To represent the evolution between locations of the machine when it is reading a word, the letters forming the timed state sequence alternate between locations of S and actions of A. Nevertheless, we need these actions to appear instantaneously (to enforce the channels to be FIFO, as we will see in condition 4.), so that we make a supplementary letter n (for 'Nothing') appear in an open interval after each $s \in S$ and $a \in A$:
$$\Phi_{alt} \equiv \Box \left(\bigwedge_{s \in S} \left(s \Rightarrow \left(n\, U \bigvee_{a \in A} a \right) \right) \wedge \bigwedge_{a \in A} \left(a \Rightarrow \left(n\, U \bigvee_{s \in S} s \right) \right) \right).$$
3. Omitting the n's present in the timed state sequence, each triple (s, a, s') of consecutive letters of the timed state sequence starting with a location

[9] As on the pointwise semantics, it is easy to adapt this proof for the unstrict semantics of the U operator, using atomic propositions instead of simple letters.

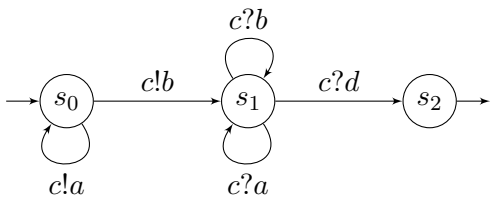


Figure 2.34: Channel machine $\mathcal{C}$

$s \in S$ must be a triple of Δ:

$$\Phi_\Delta \equiv \Box \left(\bigwedge_{s \in S} \left(s \Rightarrow \left(\bigvee_{(s,a,s') \in \Delta} \left(n\, U \left(a \wedge \left(n\, U s' \right) \right) \right) \right) \right) \right).$$

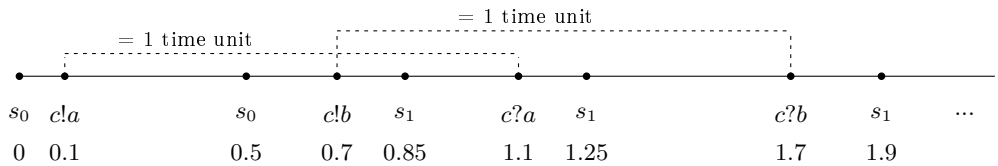
4. To force the channels to be FIFO, we furthermore require that each action of type $c!m$ is followed exactly one time unit later by the corresponding action $c?m$ and that each action of type $c?m$ is preceded exactly one time unit earlier by the corresponding action $c!m$:

$$\Phi_{FIFO} \equiv \Box \left(\left(\bigwedge_{c!m \in A} \left(c!m \Rightarrow \Diamond_{[1,1]} c?m \right) \right) \wedge \left(\bigwedge_{c?m \in A} \left(\left(\Diamond_{[1,1]} c?m \right) \Rightarrow c!m \right) \right) \right).$$

Example 2.175. Figure 2.34 gives a perfect channel machine $\mathcal{C} = (S, s_0, M, C, \Delta)$ with a unique channel c and $M = \{a, b, d\}$. Figure 2.35 gives the encoding of a run of $\mathcal{C}$ as a timed state sequence: for the sake of readability, we do not represent the occurrences of letter 'n' on the picture.

Using this encoding, it is straightforward to reduce the halting problem (for finite runs) of a channel machine to the MTL satisfiability problem on finite timed state sequences, with the continuous semantics. Moreover, this satisfiability problem can be easily reduced to the corresponding MTL model-checking problem, giving the excepted result.

Table 2.1 summarizes the decidability results for the MTL model-checking problems with the pointwise as well as the continuous semantics. It shows that,



$(s_0, [0,0])$ $(n,]0, 0.1[)$ $(c!a, [0.1, 0.1])$ $(n,]0.1, 0.5[)$ $(s_0, [0.5, 0.5])$ $(n,]0.5, 0.7[)$ $(c!b, [0.7, 0.7])$ $(n,]0.7, 0.85[)$ $(s_1, [0.85, 0.85])$ $(n,]0.85, 1.1[)$ $(c?a, [1.1, 1.1])$ $(n,]1.1, 1.25[)$ $(s_1, [1.25, 1.25])$ $(n,]1.25, 1.7[)$ $(c?b, [1.7, 1.7])$ $(n,]1.7, 1.9[)$ $(s_1, [1.9, 1.9])$ ...

Figure 2.35: Encoding of a run of $\mathcal{C}$

Pointwise Semantics		Continuous semantics	
Finite words	Infinite words	Finite words	Infinite Words
Undecidable [7]	Undecidable [7]	Decidable (non prim. rec.) [51]	Undecidable [50]

Table 2.1: Summary of the decidability results for the MTL model-checking problems

in its unique decidable version, the MTL model-checking problem has a non-primitive recursive complexity. This is an obstacle to the creation of an MTL model-checking algorithm efficient in practice. In the following subsection, we hence consider some alternative real-time logics to MTL whose model-checking theoretical complexity is more engaging.

2.11 Alternative real-time logics

Because of the undecidability results about the MTL model-checking problems, several alternative real-time logics have been considered. The aim was to recover a model-checking problem decidable with a complexity low enough

to consider the elaboration of an algorithm applicable in practice. We start by presenting the syntactic fragment of MTL called Metric Interval Temporal Logic (MITL), that has been proposed by Alur *and al.* [5]. We then consider its fragment $\text{MITL}_{0,\infty}$. Finally, we define the Event-Clock Logic (ECL), and the fragment $\text{LTL}_{\lhd}$ of this logic.

2.11.1 MITL

MITL is a syntactic fragment of MTL where singular intervals are disallowed on the modalities. Thanks to this restriction, MITL model-checking is EXPSPACE-complete, even on infinite words: MITL thus seems a good compromise between *expressiveness* and *complexity*. Here is the formal definition of MITL ; we also recall the result from [5] giving the class of complexity of the MITL model-checking problem.

Definition 2.176. *MITL is the syntactic fragment of MTL that consists of all formulas Φ such that each interval I appearing in Φ is non-singular.*

Example 2.177. The MTL formula $\Phi_1 \equiv \Box(p \Rightarrow \Diamond_{[2,3]}q)$ is also an MITL formula because the intervals $[0, +\infty[$ (associated with operator $\Box$) and $[2, 3]$ are not singular.
In contrary, $\Phi_2 \equiv \Box(p \Rightarrow \Diamond_{[2,2]}q)$ is still an MTL formula but not an MITL formula because of the singular interval $[2, 2]$.

One can check in [50] that the proof of undecidability of the MTL model-checking problem is strongly linked to the use of singular intervals. In fact, Alur *and al.* proved that the MITL satisfiability and model-checking problems are decidable.

Theorem 2.178 ([5])**.** *On the continuous semantics, over infinite words, the MITL satisfiability and model-checking problems are* EXPSPACE-*complete.*

To the best of our knowledge, there exists no self-contained proof that the

MITL satisfiability and model-checking problems are EXPSPACE-complete over the pointwise semantics (for finite as well as infinite words).

In their seminal work, Alur *and al.* provide a construction to translate an MITL formula Φ into a timed automaton $\mathcal{B}_\Phi$, from which the automaton-based model checking procedure can be applied. Although this procedure is foundational from the theoretical point of view, it does not seem easily amenable to efficient implementation: the construction is quite involved, and requests that $\mathcal{B}_\Phi$ be completely built before the synchronous product with the system's model can be explored. Among the contributions of this thesis, we provide a translation, from each MITL formula Φ to a timed automaton $\mathcal{B}_\Phi^{pw}$ recognizing $[\![\Phi]\!]$ over the pointwise semantics. This construction is more intuitive than that presented in [5]. Moreover, we will present algorithms enabling to solve the MITL model-checking problems over finite and infinite words without construting *a priori* the entire automaton $\mathcal{B}_\Phi^{pw}$.

2.11.2 MITL$_{0,\infty}$

MITL$_{0,\infty}$ has been pointed out in [5] as a 'PSPACE fragment' of MITL, according to its model-checking problem on the continuous semantics. Later, an attractive result of [38, 60], showed that MITL$_{0,\infty}$ has the same expressiveness as MITL on the continuous semantics. However, on the pointwise semantics, MITL$_{0,\infty}$ is strictly less expressive than MITL.

Definition 2.179. *MITL$_{0,\infty}$ is the syntactic fragment of MITL that consists of all formulas Φ such that, for each interval I appearing in Φ, either* $\inf(I) = 0$ *or* $\sup(I) = +\infty$.

Theorem 2.180 ([5])**.** *On infinite words, over the continuous semantics, the MITL$_{0,\infty}$ model-checking problem is PSPACE-complete.*

Theorem 2.181 ([38, 56, 60])**.** *On the continuous semantics, over infinite words, MITL$_{0,\infty}$ has the same expressive power as MITL.*

Theorem 2.182 ([56]). *On the pointwise semantics, over infinite words, MITL$_{0,\infty}$ is strictly less expressive than MITL.*

2.11.3 ECL

The Event-Clock Logic (ECL) has been introduced in [57]. In this subsection, we recall the pointwise semantics of ECL. For the sake of completeness, we also explain the continuous semantics of ECL, while we do not have any contribution using it. Finally, we present some results about the relative expressiveness of ECL, MITL and MITL$_{0,\infty}$, according to the considered semantics. Remark that the definitions of ECL is based on *atomic propositions* instead of on an alphabet.

Definition 2.183 (Pointwise syntax of ECL). *Given a set AP of atomic propositions, the formulas of ECL over the pointwise semantics are defined by the following syntax, where $p \in AP$, $c \in \mathbb{N}$ and $\sim \in \{<, \leqslant, =, >, \geqslant\}$:*

$$\Phi \ := \ p \mid \Phi_1 \vee \Phi_2 \mid \neg\Phi \mid \bigcirc \Phi \mid \ominus \Phi \mid \Phi_1 U \Phi_2 \mid \Phi_1 S \Phi_2 \mid \rhd_{\sim c} \Phi \mid \lhd_{\sim c} \Phi.$$

This syntax uses the operators until (U), since (S), a prophecy operator ($\rhd$) and a history operator ($\lhd$).

Definition 2.184 (Pointwise semantics of ECL). *Given a timed word $\theta = (\overline{\sigma}, \overline{\tau})$ over the set AP of atomic propositions, a position $1 \leqslant i \leqslant |\theta|$ and an ECL formula Φ, we say that θ satisfies Φ from position i, written $(\theta, i) \models \Phi$ iff the following holds:*

- $(\theta, i) \models \sigma$ *iff* $p \in \sigma_i$,
- $(\theta, i) \models \Phi_1 \vee \Phi_2$ *iff* $(\theta, i) \models \Phi_1$ *or* $(\theta, i) \models \Phi_2$,
- $(\theta, i) \models \neg\Phi$ *iff* $(\theta, i) \not\models \Phi$,
- $(\theta, i) \models \bigcirc\Phi$ *iff* $(\theta, i+1) \models \Phi$,

- $(\theta, i) \models \ominus\Phi$ *iff* $i > 0$ *and* $(\theta, i-1) \models \Phi$,
- $(\theta, i) \models \Phi_1 U \Phi_2$ *iff* $\exists i \leqslant j \leqslant |\theta|$, *such that* $(\theta, j) \models \Phi_2$ *and* $\forall i \leqslant k < j, (\theta, k) \models \Phi_1$,
- $(\theta, i) \models \Phi_1 S \Phi_2$ *iff* $\exists 0 \leqslant j \leqslant i$, *such that* $(\theta, j) \models \Phi_2$ *and* $\forall j < k \leqslant i, (\theta, k) \models \Phi_1$,
- $(\theta, i) \models \rhd_{\sim c}\Phi$ *iff* $\exists i < j \leqslant |\theta|$, *such that* $(\theta, j) \models \Phi$ *and* $\forall i < k < j, (\theta, k) \not\models \Phi$ *and* $\tau_j - \tau_i \sim c$,
- $(\theta, i) \models \lhd_{\sim c}\Phi$ *iff* $\exists 0 \leqslant j < i$, *such that* $(\theta, j) \models \Phi$ *and* $\forall j < k < i, (\theta, k) \not\models \Phi$ *and* $\tau_i - \tau_j \sim c$.

We say that θ satisfies Φ, *written* $\theta \models \Phi$, *iff* $(\theta, 1) \models \Phi$. *We note* $[\![\Phi]\!] = \{\theta \mid |\theta| < \infty$ *and* $\theta \models \Phi\}$ *the* finite timed language *of* Φ *and* $[\![\Phi]\!]^\omega = \{\theta \mid |\theta| = +\infty$ *and* $\theta \models \Phi\}$ *the* infinite timed language *of* Φ.

Example 2.185. The ECL formula $\Box(a \Rightarrow \rhd_{\leqslant 5} b)$ means that each a is followed by a b within 5 time units.
$a \wedge \Box(a \Rightarrow ((\rhd_{\leqslant 1} a) \wedge (\rhd_{\geqslant 1} a)))$ is also an ECL formula and means that a is read exactly in every integer time unit.
The ECL formula $\Box(((\lhd_{\leqslant 3} a) \wedge (\lhd_{\geqslant 3} a)) \Rightarrow b)$ means that, if the last a was read exactly 3 time units ago, then a b must be read now.
Remark that the MTL formula $\Phi_{MTL} \equiv \Box(a \Rightarrow \Diamond_{[1,1]} b)$ and the ECL formula $\Phi_{ECL} \equiv \Box(a \Rightarrow ((\rhd_{\leqslant 1} b) \wedge (\rhd_{\geqslant 1} b)))$ have different meanings. Φ_{MTL} says that each a is followed by a b exactly one time unit later, while Φ_{ECL} is stronger and furthermore requires that the *first* following b is at exactly 1 time unit.

For the sake of completeness, we also present the continuous syntax and semantics of ECL.

Definition 2.186 (Continuous syntax of ECL)**.** *Given a set AP of atomic propositions, the formulas of ECL over the continuous semantics are defined by the following syntax, where* $p \in AP$ *and* $I \in \mathcal{I}(\mathbb{N}^\infty)$*):*

$$\Phi \ := \ p \mid \Phi_1 \vee \Phi_2 \mid \neg\Phi \mid \Phi_1 U \Phi_2 \mid \Phi_1 S \Phi_2 \mid \rhd_I \Phi \mid \lhd_I \Phi.$$

Definition 2.187 (Continuous semantics of ECL). *Given a timed state sequence $\Theta = (\overline{\sigma}, \overline{I})$ over the set AP of atomic propositions and an ECL formula Φ, we say that Θ satisfies Φ at time $t \in \mathbb{R}^+$, written $(\Theta, t) \models \Phi$, iff the following holds:*

- $(\Theta, t) \models \sigma$ *iff there exists* $i \geqslant 1$ *and* $I_i \in \overline{I}$ *such that* $t \in I_i$ *and* $p \in \sigma_i$,
- $(\Theta, t) \models \Phi_1 \vee \Phi_2$ *iff* $(\Theta, t) \models \Phi_1$ *or* $(\Theta, t) \models \Phi_2$,
- $(\Theta, t) \models \neg\Phi$ *iff* $(\Theta, t) \not\models \Phi$,
- $(\Theta, t) \models \Phi_1 U \Phi_2$ *iff* $\exists t' > t$ *with* $(\Theta, t') \models \Phi_2$ *and* $\forall t'' \in]t, t'[, (\Theta, t'') \models \Phi_1 \vee \Phi_2$,
- $(\Theta, t) \models \Phi_1 S \Phi_2$ *iff* $\exists t' < t$ *with* $(\Theta, t') \models \Phi_2$ *and* $\forall t'' \in]t', t[, (\Theta, t'') \models \Phi_1 \vee \Phi_2$,
- $(\Theta, t) \models \rhd_J \Phi$ *iff* $\exists t' > t$ *with* $t' \in (J + t), (\Theta, t') \models \Phi$ *and* $\forall t'' > t$ *with* $t'' < (J + t),\ (\Theta, t'') \not\models \Phi$,
- $(\Theta, t) \models \lhd_J \Phi$ *iff* $\exists t' < t$ *with* $t' \in (t - J), (\Theta, t') \models \Phi$ *and* $\forall t'' < t$ *with* $t'' > (t - J),\ (\Theta, t'') \not\models \Phi$.

We say that Θ satisfies Φ, *written* $\Theta \models \Phi$, *iff* $(\Theta, 0) \models \Phi$. *We note* $[\![\Phi]\!]_{cont} = \{\Theta \mid |\Theta| < \infty$ *and* $\Theta \models \Phi\}$ *the* finite timed state sequence language *of* Φ *and* $[\![\Phi]\!]^{\omega}_{cont} = \{\Theta \mid |\Theta| = +\infty$ *and* $\Theta \models \Phi\}$ *the* infinite timed state sequence language *of* Φ.

Remark 2.188. We notice that the definition of the U operator differs from the one given on the pointwise semantics. This choice was made (in [57]) to keep a definition of formulas satisfaction close to intuition. For instance, let us consider formulas Φ_1 and Φ_2 such that: (i) on the interval $[0, 1]$, Φ_1 is satisfied and Φ_2 is not satisfied, and (ii) on the interval $]1, +\infty[$, Φ_1 is not satisfied and Φ_2 is satisfied. The intuition we have of the definition of $\Phi_1 U \Phi_2$ makes us think that

this formula should be satisfied. With the present definition, it is indeed the case. If we consider a definition imitating that of the pointwise semantics, as

$$(\Theta, t) \models \Phi_1 U \Phi_2 \text{ iff } \exists t' > t \text{ with } (\Theta, t') \models \Phi_2 \text{ and } \forall t'' \in]t, t'[, (\Theta, t'') \models \Phi_1,$$

then, $\Phi_1 U \Phi_2$ is not satisfied anymore. Indeed, this definition requires there is a *first* instant t' in which formula Φ_2 is satisfied. Such an instant does not exists in this example.

Here are some results about the expressiveness of ECL, according to the semantics we consider. These results are proven in the given references, for MITL and $\text{MITL}_{0,\infty}$ using the *strict* semantics of the U operator and an additional operator 'since' (S): we will respectively note them MITL^S and $\text{MITL}^S_{0,\infty}$.

Theorem 2.189. *[56, 57] On the pointwise semantics, over infinite words, ECL and $MITL^S_{0,\infty}$ are equally expressive, and $MITL^S_{0,\infty}$ is strictly less expressive than $MITL^S$.*

Theorem 2.190. *[56, 57] On the continuous semantics over infinite words, ECL, $MITL^S_{0,\infty}$ and $MITL^S$ are equally expressive.*

As for MTL and its fragments, the satisfiability problem of ECL was investigated. It was proved to be PSPACE-complete. The proof relies on the use of another kind of automata, called Event-clock automata, in way to represent the ECL timed languages. These automata have good properties, in particular, they are seen as a 'determinizable class of timed automata' ([6, 56]).

Theorem 2.191. *[56, 57] The ECL satisfiability problem is PSPACE-complete, on the pointwise and the continuous semantics, over infinite words.*

Finally, the ECL reactive synthesis problem (RS, see Definition 2.127) has been examined in [28], over infinite words with the pointwise semantics. The authors proved that this problem is undecidable. One can see that we rewrote the definition given in [28]. However, it is easy to see that our version of the definition of reactive synthesis is equivalent to the one presented in this paper.

Theorem 2.192 ([28]). *The ECL RS problem is undecidable, when ECL is interpreted over infinite words, with the pointwise semantics.*

As MITL^S is more expressive than ECL, the next corollary directly follows this theorem.

Corollary 2.193. *The MITL^S RS problem is undecidable, when MITL^S is interpreted over infinite words, with the pointwise semantics.*

2.11.4 LTL$_{\lhd}$

The fragment LTL$_{\lhd}$ of ECL has been pointed out in [28], treating of the reactive synthesis problem. We only define and use it on the pointwise semantics.

Definition 2.194. *LTL$_{\lhd}$ is the (pointwise) syntactic fragment of ECL that consists of all formulas Φ that do not contain any operator $\ominus$ nor $\rhd_I$.*

The LTL$_{\lhd}$ reactive synthesis problem (RS, see Definition 2.127) was also examined in [28]. The authors proved that this problem is decidable, over infinite words with the pointwise semantics:

Theorem 2.195 ([28]). *The LTL$_{\lhd}$ RS problem is 2EXPTIME-complete, when LTL$_{\lhd}$ is interpreted over infinite words, with the pointwise semantics.*

Figure 2.11.4 summarizes the results about the relative expressiveness of the logics defined in the previous subsections over infinite words. The left picture concerns the pointwise semantics and the right one concerns the continuous semantics.

Figure 2.36: Relative expressiveness over infinite words for the pointwise semantics (left) and the continuous semantics (right).

Part I

MITL Satisfiability and Model-Checking

CHAPTER 3

An interval semantics for OCATA

In Chapter 2, we fixed notations and recalled the notions of timed automata (over finite words), Büchi timed automata (over infinite words) and one-clock alternating timed automata (OCATA). We also exhibited the syntax and semantics of several logics and displayed well-known results about the (un)decidability and/or complexity of their satisfiability and model-checking problems.
The aim of the present part of the thesis is to present a new approach of the *MITL model-checking* problem, over the pointwise semantics, using OCATA. Henzinger and al. proved that the MITL model-checking problem is EXPSPACE-complete [5]. They provide an MITL model-checking algorithm based on a very complex and unintuitive construction (5 pages) of Büchi timed automata for MITL formulas. Our goal is to revisit the MITL model-checking and provide new algorithms on finite and infinite words with the pointwise semantics: we hope these algorithms are rather intuitive. These algorithms are inspired by one of the most efficient techniques, to a practical point of view, for the *LTL model-checking* problem. It consists in translating the LTL formula into an alternating automaton before to change it into a Büchi automaton [45]. We follow the same steps using

one clock alternating timed automata (OCATA) and (Büchi) timed automata. Nevertheless, to simplify the translation of the constructed OCATA to (Büchi) timed automata, we begin by defining a new semantics of these OCATA. This is the object of the present chapter. The contributions present in this chapter constitute a part of our publication [19] (arXiv reference: [20]).

The *standard* semantics for OCATA [42, 49] is defined as an infinite transition system whose *configurations* are finite sets of pairs (ℓ, v), where ℓ is a location and v is the valuation of the (unique) clock (see Section 2.7 for details). In this chapter, we introduce a *novel* semantics for OCATA, in which *configurations* are sets of *states* (ℓ, I), where ℓ is a location of the OCATA and I is an *interval*, instead of a single point in $\mathbb{R}^+$. Intuitively, a state (ℓ, I) is an abstraction of all the states (ℓ, v) with $v \in I$, in the standard semantics. We further introduce the notion of *approximation function*. Roughly speaking, an approximation function associates with each configuration C (in the interval semantics), a set of configurations that *approximates* C (in a sense that will be made precise later), *and contains less states* than C. In Section 5.1, we will show that the interval semantics, combined to a proper approximation function, allows us to build, from all MITL formula Φ, an OCATA $\mathcal{A}_\Phi$ accepting $[\![\Phi]\!]$, and whose *reachable configurations contain a bounded number of intervals*. This will be the basis of our algorithm to build a *timed automaton* recognising Φ (and hence performing automata-based model-checking of MITL).

Our definition of the *interval semantics* for OCATA follows the definition of the classical semantics as given by Ouaknine and Worrell [49] (see Section 2.7), adapted to cope with intervals. Hence, we will abusively use the same vocabulary for our new semantics than for the classical one. For example, despite we will now use 'interval states' of the form (ℓ, I), instead of the states (ℓ, v) of the classical semantics, we will abusively simply call (ℓ, I) a *state*.

Interval semantics. We will now define the interval semantics of OCATA. We start by defining what is a *state* and a *configuration* for this novel semantics. Then, we show how an OCATA can pass from one configuration to another using

a new notion of *minimal model*.

Definition 3.1. *We call* state *of an OCATA* $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ *a couple* (ℓ, I) *where* $\ell \in L$ *and* $I \in \mathcal{I}(\mathbb{R}^+)$. *A state* (ℓ, I) *is* accepting *iff* $\ell \in F$.
We note $S = L \times \mathcal{I}(\mathbb{R}^+)$ *the state space of* $\mathcal{A}$.

When $I = [v, v]$ (sometimes denoted $I = \{v\}$), we shorten (ℓ, I) by (ℓ, v).

Definition 3.2. *A* configuration *of an OCATA* $\mathcal{A}$ *is a (possibly empty) finite set of states of* $\mathcal{A}$ *whose intervals associated with a same location are disjoint. The* initial configuration *of* $\mathcal{A}$ *is* $\{(\ell_0, [0, 0])\}$. *A configuration is* accepting *iff all the states it contains are accepting (in particular, the empty configuration is accepting).*
For a configuration C *and a delay* $t \in \mathbb{R}^+$, *we note* $C + t$ *the configuration* $\{(\ell, I + t) | (\ell, I) \in C\}$.
We note $\mathrm{Config}(\mathcal{A})$ *the set of all configurations of* $\mathcal{A}$.

In the rest of this thesis, we sometimes see a configuration C as a function from L to $2^{\mathcal{I}(\mathbb{R}^+)}$ such that for all $\ell \in L$: $C(\ell) = \{I \mid (\ell, I) \in C\}$. From now on, we assume that, for all configurations C and all locations ℓ: when writing $C(\ell)$ as $\{I_1, \ldots, I_m\}$ we have $I_i < I_{i+1}$ for all $1 \leqslant i < m$.

Remark 3.3. When considering two configurations of an OCATA $\mathcal{A}$, their union might not be a configuration of $\mathcal{A}$. For instance, consider the OCATA of Figure 3.1 and its configurations $C = \{(\ell_0, [0, 1.5])\}$ and $D = \{(\ell_0, [0.5, 1.8])\}$, $C \cup D = \{(\ell_0, [0, 1.5]), (\ell_0, [0.5, 1.8])\}$ is not a configuration because the two intervals associated with location ℓ_0 are not disjoint. Intuitively, we would like that $C \cup D = \{(\ell_0, [0, 1.8])\}$.
To get round this problem, from a finite set of states S, we use the following procedure to construct a *well defined configuration*, denoted $S^{\neq}$, representing S:

1. $A^0 = S$,

2. for $i \geqslant 0$, while, for a certain $\ell \in L$, there are (ℓ, I) and (ℓ, J) such that $I \cap J \neq \varnothing$ in A^i, $A^{i+1} = \big(A^i \backslash \{(\ell, I), (\ell, J)\}\big) \cup \{(\ell, I \cup J)\}$.

As A^0 is finite, there is a smallest $i^\star \geqslant 0$ such that $\forall j \geqslant i^\star$, $A^j = A^{i^\star}$. We note $A^\star = A^{i^\star}$. $A^\star$ is the configuration $S^{\neq}$ we are looking for.
Thanks to this procedure, we can represent the union of *configurations* C and D by the *configuration* $(C \cup D)^{\neq}$.

We now present the notion of *number of clock copies* present in a configuration. It will play a key role in the main results of this thesis. Intuitively, being given a finite set of intervals E, the *number of clock copies* of E is the number of *individual clocks* we need to encode all the information present in E, using one clock to track singular intervals, and two clocks to retain $\inf(I)$ and $\sup(I)$ respectively for non-singular intervals I.

Definition 3.4. *Let E be a finite set of intervals from $\mathcal{I}(\mathbb{R}^+)$. We let*

$$\|E\| \;=\; |\{[a,a] \in E\}| \;+\; 2 \times |\{I \in E \mid \inf(I) \neq \sup(I)\}|$$

denote the number of clock copies *of E.*
For a configuration C, we let:

$$\|C\| = \sum_{\ell \in L} \|C(\ell)\| \,.$$

Example 3.5. Let us consider the OCATA $\mathcal{A}$ of Figure 3.1. $(\ell_1, [0.3, 2])$ is a state of $\mathcal{A}$, as well as $(\ell_0, [2, 2])$, which could be shortly denoted $(\ell_0, 2)$. $C = \{(\ell_1, [0.3, 2]), (\ell_1, [0.1, 0.2]), (\ell_0, [2, 2])\}$ is a configuration of $\mathcal{A}$. We can see C as a function from $\{\ell_0, \ell_1, \ell_2\}$ to $2^{\mathcal{I}(\mathbb{R}^+)}$ in the following way: $C(\ell_0) = \{[2, 2]\}$, $C(\ell_1) = \{[0.1, 0.2], [0.3, 2]\}$ and $C(\ell_2) = \varnothing$. The number of clock copies of C is: $\|C\| = \|C(\ell_0)\| + \|C(\ell_1)\| + \|C(\ell_2)\| \;=\; 1 + 4 + 0 \;=\; 5$.

We now define the satisfaction relation '$\models_I$' enabling to express the notion of *minimal model.*

Definition 3.6. *Let $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ be an OCATA. We consider a set of states M of $\mathcal{A}$ and $I \in \mathcal{I}(\mathbb{R}^+)$. We define the satisfaction relation '$\models_I$' on $\Gamma(L)$ as:*

- $M \models_I \top$,
- $M \models_I \gamma_1 \wedge \gamma_2$ *iff* $M \models_I \gamma_1$ *and* $M \models_I \gamma_2$,
- $M \models_I \gamma_1 \vee \gamma_2$ *iff* $M \models_I \gamma_1$ *or* $M \models_I \gamma_2$,
- $M \models_I \ell$ *iff* $(\ell, I) \in M$,
- $M \models_I x \bowtie c$ *iff* $\forall x \in I$, $x \bowtie c$,
- $M \models_I x.\gamma$ *iff* $M \models_{[0,0]} \gamma$.

Definition 3.7. *Let $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ be an OCATA. We say that $M^{\neq} \in$ Config $(\mathcal{A})$ (see Remark 3.3) is a minimal model of the formula $\gamma \in \Gamma(L)$ with respect to the interval $I \in \mathcal{I}(\mathbb{R}^+)$ iff $M \models_I \gamma$ and for all $M' \in$ Config $(\mathcal{A})$ such that $M' \subsetneq M$, $M' \not\models_I \gamma$.*

Remark that a formula γ can admit several minimal models (at most one for each disjunct in the case of a formula of the form $\gamma = \bigvee_j \bigwedge_k A_{j,k}$, see Example 3.8). Intuitively, for $\ell \in L, \sigma \in \Sigma$ and $I \in \mathcal{I}(\mathbb{R}^+)$, a minimal model of $\delta(\ell, \sigma)$ with respect to I represents a configuration the automaton can reach from state (ℓ, I) by reading σ. The definition of $M \models_I x \bowtie c$ only allows to take a transition $\delta(\ell, \sigma)$ from state (ℓ, I) if all the values in I satisfy the clock constraint $x \bowtie c$ of $\delta(\ell, \sigma)$.

Example 3.8. Let us consider again the OCATA of Figure 3.1. First, let us look for the minimal models of $\delta(\ell_1, a)$ with respect to $[0, 0.3]$. They are obtained thanks to a set of states M such that: $M \models_{[0,0.3]} (\ell_1 \wedge x \neq 1) \vee \ell_2$. As $\forall x \in [0, 0.3]$,

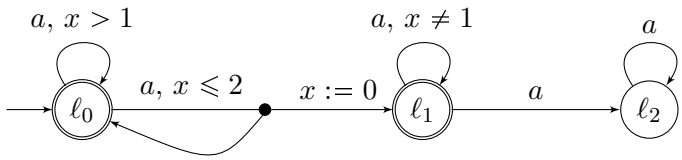


Figure 3.1: OCATA $\mathcal{A}$

$x \neq 1$, we have:

$$\begin{array}{lll} & M \models_{[0,0.3]} (\ \ell_1 \wedge x \neq 1\ ) & \vee \quad \ell_2 \\ \text{iff} & M \models_{[0,0.3]} \ell_1 & \text{or} \quad M \models_{[0,0.3]} \ell_2 \\ \text{iff} & (\ell_1, [0, 0.3]) \in M & \text{or} \quad (\ell_2, [0, 0.3]) \in M. \end{array}$$

So, $(\{(\ell_1, [0, 0.3])\})^{\neq} = \{(\ell_1, [0, 0.3])\}$ and $(\{(\ell_2, [0, 0.3])\})^{\neq} = \{(\ell_2, [0, 0.3])\}$ are the two minimal models of $\delta(\ell_1, a)$ with respect to $[0, 0.3]$.
Now, we will look for the minimal models of $\delta(\ell_0, a)$ with respect to $[0, 0.5]$. They are obtained thanks to a set of states M such that: $M \models_{[0,0.5]} (\ell_0 \wedge x > 1) \vee (\ell_0 \wedge x.\ell_1 \wedge x \leqslant 2)$. Remark that, for instance, $0 \in [0, 0.5]$ and is such that $0 \not> 1$, while $\forall x \in [0, 0.5]$, $x \leqslant 2$, we so have:

$$\begin{array}{ll} & M \models_{[0,0.5]} (\ \ell_0 \wedge x > 1\ ) \ \vee \ (\ \ell_0 \wedge x.\ell_1 \wedge x \leqslant 2\ ) \\ \text{iff} & M \models_{[0,0.5]} (\ \ell_0 \wedge x.\ell_1 \wedge x \leqslant 2\ ) \\ \text{iff} & M \models_{[0,0.5]} \ell_0 \text{ and } M \models_{[0,0.5]} x.\ell_1 \\ \text{iff} & (\ell_0, [0, 0.5]) \in M \text{ and } M \models_{[0,0]} \ell_1 \\ \text{iff} & (\ell_0, [0, 0.5]) \in M \text{ and } (\ell_1, [0, 0]) \in M. \end{array}$$

So, $(\{(\ell_0, [0, 0.5]), (\ell_1, 0)\})^{\neq} = \{(\ell_0, [0, 0.5]), (\ell_1, 0)\}$ is the unique *minimal model* of $\delta(\ell_0, a)$ with respect to $[0, 0.5]$.
Finally, let us look for the minimal models of $\delta(\ell_0, a)$ with respect to $[1.7, 1.9]$. They are obtained thanks to a set of states M such that: $M \models_{[1.7,1.9]} (\ell_0 \wedge x > 1) \vee (\ell_0 \wedge x.\ell_1 \wedge x \leqslant 2)$. As $\forall x \in [1.7, 1.9]$, $x > 1$ and $\forall x \in [1.7, 1.9]$, $x \leqslant 2$, we

have:

$$
\begin{array}{ll}
 & M \models_{[1.7,1.9]} (\ \ell_0 \wedge x > 1\ ) \ \vee\ (\ \ell_0 \wedge x.\ell_1 \wedge x \leqslant 2\ ) \\
\text{iff} & M \models_{[1.7,1.9]} \ell_0 \quad \text{or} \quad (\ M \models_{[1.7,1.9]} \ell_0 \text{ and } M \models_{[1.7,1.9]} x.\ell_1\ ) \\
\text{iff} & (\ell_0, [1.7, 1.9]) \in M \qquad \text{or} \\
 & (\ (\ell_0, [1.7, 1.9]) \in M \text{ and } M \models_{[0,0]} \ell_1\ ) \\
\text{iff} & (\ell_0, [1.7, 1.9]) \in M \qquad \text{or} \\
 & (\ (\ell_0, [1.7, 1.9]) \in M \text{ and } (\ell_1, [0, 0]) \in M\ ).
\end{array}
$$

So, $(\{(\ell_0, [1.7, 1.9])\})^{\neq}$ and $(\{(\ell_0, [1.7, 1.9]), (\ell_1, 0)\})^{\neq}$ are two models of $\delta(\ell_0, a)$ with respect to $[1.7, 1.9]$, but as $\{(\ell_0, [1.7, 1.9])\} \subseteq \{(\ell_0, [1.7, 1.9]), (\ell_1, 0)\}$, $(\{(\ell_0, [1.7, 1.9])\})^{\neq} = \{(\ell_0, [1.7, 1.9])\}$ is the unique *minimal model* of $\delta(\ell_0, a)$ with respect to $[1.7, 1.9]$.

The following example shows that the set of states M satisfying $\delta(\ell, \sigma)$ with respect to I, in the sense of Definition 3.6, might not be a configuration. This explains why we defined a *minimal model* to be $M^{\neq}$ instead of M.

Example 3.9. Let us consider the OCATA of Figure 3.2. Let us look for the minimal models of $\delta(\ell, a)$ with respect to $[0, 0.5]$. They are obtained thanks to a set of states M such that: $M \models_{[0,0.5]} (\ell \wedge x.\ell)$. We so have:

$$
\begin{array}{ll}
 & M \models_{[0,0.5]} (\ \ell \wedge x.\ell\ ) \\
\text{iff} & M \models_{[0,0.5]} \ell \ \text{ and } \ M \models_{[0,0.5]} x.\ell \\
\text{iff} & (\ell, [0, 0.5]) \in M \ \text{ and } \ M \models_{[0,0]} \ell \\
\text{iff} & (\ell, [0, 0.5]) \in M \ \text{ and } (\ell, [0, 0]) \in M.
\end{array}
$$

$\{(\ell, [0, 0.5]), (\ell, [0, 0])\}$ is a set of states that is not a configuration, while $(\{(\ell, [0, 0.5]), (\ell, [0, 0])\})^{\neq} = \{(\ell, [0, 0.5])\}$ is a configuration and the unique *minimal model* of $\delta(\ell, a)$ with respect to $[0, 0.5]$.

We now define some notations enabling to simply express what are the possible *successors* of a given configuration of an OCATA.

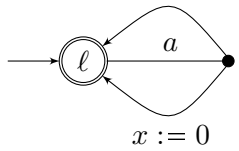


Figure 3.2: An OCATA $\mathcal{A}$ on $\Sigma = \{a\}$.

Definition 3.10. *Let $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ be an OCATA and (ℓ, I) be a state of $\mathcal{A}$. We let $\mathsf{Succ}((\ell, I), \sigma) = \{M \mid M$ is a minimal model of $\delta(\ell, \sigma)$ with respect to $I\}$. We lift the definition of Succ to configurations C as follows: $\mathsf{Succ}(C, \sigma)$ is the set of all configurations C' of the form $(\cup_{s \in C} M_s)^{\neq}$, where, for all $s \in C$: $M_s \in \mathsf{Succ}(s, \sigma)$. That is, each $C' \in \mathsf{Succ}(C, \sigma)$ is obtained by choosing one minimal model M_s in $\mathsf{Succ}(s, \sigma)$ for each $s \in C$, taking the union of all those M_s, and transform the obtained set of states of $\mathcal{A}$ in a real configuration (managing non-disjoint intervals).*

Example 3.11. Let us consider the configuration $C = \{(\ell_0, [0, 0.5]), (\ell_1, [0, 0.3])\}$ of the OCATA of Figure 3.1. We saw in the Example 3.8 that the unique minimal model of $\delta(\ell_0, a)$ with respect to $[0, 0.5]$ is $C_1 = \{(\ell_0, [0, 0.5]), (\ell_1, 0)\}$, and that the two minimal models of $\delta(\ell_1, a)$ with respect to $[0, 0.3]$ are $C_2 = \{(\ell_1, [0, 0.3])\}$ and $C_3 = \{(\ell_2, [0, 0.3])\}$. So, $\mathsf{Succ}((\ell_0, [0, 0.5]), a) = \{C_1\}$ and $\mathsf{Succ}((\ell_1, [0, 0.3]), a) = \{C_2, C_3\}$. Hence, $\mathsf{Succ}(C, a) = \{(C_1 \cup C_2)^{\neq}, (C_1 \cup C_3)^{\neq}\}$. The procedure of Remark 3.3 gives

$$\begin{aligned}(C_1 \cup C_2)^{\neq} &= (\{(\ell_0, [0, 0.5]), (\ell_1, 0), (\ell_1, [0, 0.3])\})^{\neq} \\ &= \{(\ell_0, [0, 0.5]), (\ell_1, [0, 0.3])\}.\end{aligned}$$

Similarly,

$$\begin{aligned}(C_1 \cup C_3)^{\neq} &= (\{(\ell_0, [0, 0.5]), (\ell_1, 0), (\ell_2, [0, 0.3])\})^{\neq} \\ &= \{(\ell_0, [0, 0.5]), (\ell_1, 0), (\ell_2, [0, 0.3])\}.\end{aligned}$$

So, $\mathsf{Succ}(C, a) = \{\{(\ell_0, [0, 0.5]), (\ell_1, [0, 0.3])\}, \{(\ell_0, [0, 0.5]), (\ell_1, 0), (\ell_2, [0, 0.3])\}\}$.

Approximation functions. As stated before, our goal is to define a semantics for OCATA that enables to bound the number of clock copies. To this end,

we define the notion of approximation function: we will use such functions to reduce the number of clock copies associated with each location in a configuration. An approximation function associates with each configuration C a set of configurations C' such that $\|C'(\ell)\| \leqslant \|C(\ell)\|$ and such that the intervals in $C'(\ell)$, *cover* those of $C(\ell)$, for all ℓ. Then, we define the semantics of an OCATA $\mathcal{A}$ by means of a transition system $\mathsf{TTS}\,(\mathcal{A}, f)$ whose definition is *parametrised by an approximation function* f.

Definition 3.12. *Let $\mathcal{A}$ be an OCATA. An* approximation function *is a function $f : \mathrm{Config}\,(\mathcal{A}) \mapsto 2^{\mathrm{Config}(\mathcal{A})}$ such that for all configurations C, for all $C' \in f(C)$, for all locations $\ell \in L$:*

- $\|C'(\ell)\| \leqslant \|C(\ell)\|$,
- *for all $I \in C(\ell)$, there exists $J \in C'(\ell)$ such that $I \subseteq J$,*
- *for all $J \in C'(\ell)$, there are $I_1, I_2 \in C(\ell)$ such that $\inf(J) = \inf(I_1)$ and $\sup(J) = \sup(I_2)$.*

We note $APP_{\mathcal{A}}$ the set of approximation functions for $\mathcal{A}$.
We lift all approximation functions f to sets $\mathcal{C}$ of configurations in the usual way: $f(\mathcal{C}) = \cup_{C \in \mathcal{C}} f(C)$.

In the sequel, we will often use the particular approximation function $Id : \mathrm{Config}\,(\mathcal{A}) \mapsto 2^{\mathrm{Config}(\mathcal{A})}$ such that $Id(C) = \{C\}$ for all C.

Example 3.13. Let us consider an OCATA $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$. For all $C \in \mathrm{Config}\,(\mathcal{A})$ and all $\ell \in L$, let us note $C(\ell) = \{I_1^\ell, \ldots, I_{m^\ell}^\ell\}$. Then, the function $f^* : \mathrm{Config}\,(\mathcal{A}) \mapsto 2^{\mathrm{Config}(\mathcal{A})}$ such that, for all $C \in \mathrm{Config}\,(\mathcal{A})$, $f^*(C) = \bigcup_{\ell \in L}(\ell, [inf(I_1^\ell), sup(I_{m^\ell}^\ell)])$ is an approximation function.

In the rest of this thesis we will rely mainly on approximation functions that enable to *bound* the number of clock copies in all configurations along all runs of an OCATA $\mathcal{A}$:

Definition 3.14. *Let $k \in \mathbb{N}$, we say that $f \in APP_{\mathcal{A}}$ is a* k-bounded approximation function *iff for all $C \in \mathrm{Config}(\mathcal{A})$, for all $C' \in f(C)$: $\|C'\| \leqslant k$.*

In particular, the function Id is not a k-bounded approximation function.

Example 3.15. Let us consider an OCATA $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ and the approximation function f^* of Example 3.13. f^* is a k-bounded approximation function for all natural number $k \geqslant 2.|L|$ and is *not* k-bounded for natural numbers $0 \leqslant k < 2.|L|$.

f-runs of OCATA. We can now define formally the notion of *run* of an OCATA in the interval semantics. This notion will be parametrised by an approximation function f, that will be used to reduce the number of states present in each configuration along the run. Each new configuration in the run is thus obtained in three steps: letting time elapse, performing a discrete step, and applying the approximation function. These steps characterize the *f-semantics* of $\mathcal{A}$ on which is based the definition of *f-run* of $\mathcal{A}$

Definition 3.16. *Let $\mathcal{A}$ be an OCATA and let $f \in APP_{\mathcal{A}}$ be an approximation function. The* f-semantics *of $\mathcal{A}$ is the transition system $\mathsf{TTS}(\mathcal{A}, f) = (\mathrm{Config}(\mathcal{A}), \rightsquigarrow, \longrightarrow_f)$ on configurations of $\mathcal{A}$ defined as follows:*

- *the transition relation $\rightsquigarrow$ takes care of the elapsing of time: $\forall t \in \mathbb{R}^+$, $C \overset{t}{\rightsquigarrow} C'$ iff $C' = C + t$. We let $\rightsquigarrow = \bigcup_{t \in \mathbb{R}^+} \overset{t}{\rightsquigarrow}$.*
- *the transition relation $\longrightarrow_f$ takes care of discrete transitions between locations and of the approximation: $C \xrightarrow{\sigma}_f C'$ iff $C' \in f(\mathsf{Succ}(C, \sigma))$. We let $\longrightarrow_f = \bigcup_{\sigma \in \Sigma} \xrightarrow{\sigma}_f$.*

Definition 3.17. *Let $\mathcal{A}$ be an OCATA of state space S and let $f \in APP_{\mathcal{A}}$ be an approximation function. Let $\theta = (\overline{\sigma}, \overline{\tau})$ be a (finite or infinite) timed word and let us note $t_i = \tau_i - \tau_{i-1}$ for all $1 \leqslant i \leqslant |\theta|$, assuming $\tau_0 = 0$. An* f-run *of $\mathcal{A}$ on θ is a (finite or infinite) sequence of discrete and continuous transitions*

in $\mathsf{TTS}(\mathcal{A}, f)$ *that is labelled by* θ*, i.e. a sequence of the form:* $C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1}_f C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2}_f C_4 \ldots$.

In the rest of this thesis, we (sometimes) use the abbreviation $C_i \xrightarrow{t,\sigma}_f C_{i+2}$ for $C_i \overset{t}{\rightsquigarrow} C_{i+1} = C_i + t \xrightarrow{\sigma}_f C_{i+2}$.

Observe that for all pairs of configurations C, C' such that $C' \in f(\mathsf{Succ}(C + t, \sigma))$ for some f, t and σ, each $s \in C$ can be associated with a unique set $\mathsf{dest}(C, C', s) \subseteq C'$ containing all the 'successors' of s in C' and obtained as follows. Let $\overline{C} \in \mathsf{Succ}(C + t, \sigma)$ be such that $C' \in f(\overline{C})$. Thus, by definition, $\overline{C} = (\bigcup_{\overline{s} \in C} M_{\overline{s}})^{\neq}$, where each $M_{\overline{s}} \in \mathsf{Succ}(\overline{s}, \sigma)$ is the minimal model that has been chosen for $\overline{s}$ when computing $\mathsf{Succ}(C + t, \sigma)$. Then, $\mathsf{dest}(C, C', s) = \{(\ell', J) \in C' \mid (\ell', I) \in M_s \text{ and } I \subseteq J\}$. Remark that $\mathsf{dest}(C, C', s)$ is correctly defined because intervals are assumed to be disjoint in configurations.

The function dest allows to define a DAG representation of runs, as is usual with alternating automata. We regard a run π as a rooted DAG $G_\pi = (V, \rightarrow)$, whose vertices V correspond to the states of the OCATA (vertices at depth i correspond to C_{2i}), and whose set of edges $\rightarrow$ expresses the OCATA transitions. Here is its formal definition.

Definition 3.18. *Let* $\mathcal{A}$ *be an OCATA and* $\pi = C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1}_f C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2}_f \ldots \overset{t_n}{\rightsquigarrow} C_{2n-1} \xrightarrow{\sigma_n}_f C_{2n} \ldots$ *be a run of* $\mathcal{A}$*. We define the rooted DAG* $G_\pi = (V, \rightarrow)$ *with:*

- $V = \bigcup_{0 \leqslant i \leqslant |\theta|} V_i$*, where for all* $0 \leqslant i \leqslant |\theta|$*:* $V_i = \{(s, i) \mid s \in C_{2i}\}$ *is the set of all vertices of depth* i *;*
- *the root of* G_π *is* $((\ell_0, 0), 0)$ *; and*
- $(s_1, i_1) \rightarrow (s_2, i_2)$ *iff* $i_2 = i_1 + 1$ *and* $s_2 \in \mathsf{dest}(C_{2i_1}, C_{2i_2}, s_1)$.

Example 3.19. Figure 3.4 displays three DAG representation of run prefixes of $\mathcal{A}_\Phi$, the OCATA of Figure 3.3, on the timed word

$$\theta = (a, 0.1)(a, 0.2)(a, 1.9)(b, 2)(b, 3) \ldots ,$$

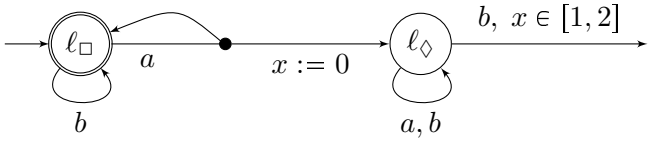


Figure 3.3: OCATA $\mathcal{A}_\Phi$ with $\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,2]} b)$.

gray boxes highlight the successive configurations. π is the Id-run prefix. It is exactly the run prefix of $\mathcal{A}_\Phi$ on θ obtained using the classical semantics.
The branches of these DAGs of the run Definition 3.18 are obtained observing function dest. For instance, let us observe the third configuration of π', say $C = \{(\ell_\Box, 0.2), (\ell_\Diamond, [0, 0.1])\}$ and its fourth configuration, say $D = \{(\ell_\Box, 1.9), (\ell_\Diamond, [0, 1.8])\}$. Let us furthermore call f the approximation function used in π', which will be so called an 'f-run'. The branches linking locations of C to locations of D are obtained thanks to $\mathsf{dest}(C, D, (\ell_\Box, 0.2))$ and $\mathsf{dest}(C, D, (\ell_\Diamond, [0, 0.1]))$. As D is computed from $C + 1.7 = \{(\ell_\Box, 1.9), (\ell_\Diamond, [1.7, 1.8])\}$ reading an a, it consists in the minimal models of $\delta(\ell_\Box, a)$ with respect to 1.9, and of $\delta(\ell_\Diamond, a)$ with respect to $[1.7, 1.8]$. In fact, $D = f\left((\bigcup_{s \in C} M_s)^{\neq}\right)$, with $M_{(\ell_\Box, 0.2)} = \{(\ell_\Box, 1.9), (\ell_\Diamond, [0, 0])\}$ and $M_{(\ell_\Diamond, [0, 0.1])} = \{(\ell_\Diamond, [1.7, 1.8])\}$. Then, the approximation function f 'merges' the intervals $[0, 0]$ and $[1.7, 1.8]$ associated with location $\ell_\Diamond$, in way D is $\{(\ell_\Box, 1.9), (\ell_\Diamond, [0, 1.8])\}$. As $[1.7, 1.8] \subseteq [0, 1.8]$, $\mathsf{dest}(C, D, (\ell_\Box, 0.2)) = \{(\ell_\Box, 1.9), (\ell_\Diamond, [0, 1.8])\}$: state $(\ell_\Box, 0.2)$ of C is linked to states $(\ell_\Box, 1.9)$ and $(\ell_\Diamond, [0, 1.8])$ of D. In a similar way, $[0, 0] \subseteq [0, 1.8]$ and hence $\mathsf{dest}(C, D, (\ell_\Diamond, [0, 0.1])) = \{(\ell_\Diamond, [0, 1.8])\}$: state $(\ell_\Diamond, [0, 0.1])$ of C is linked to state $(\ell_\Diamond, [0, 1.8])$ of D.

f-language of OCATA. We can now define the accepted language, parametrised by an approximation function f, of an OCATA, over finite and infinite timed words.

For finite timed words, the characterisation of f-runs given by Definition 3.17 is more convenient:

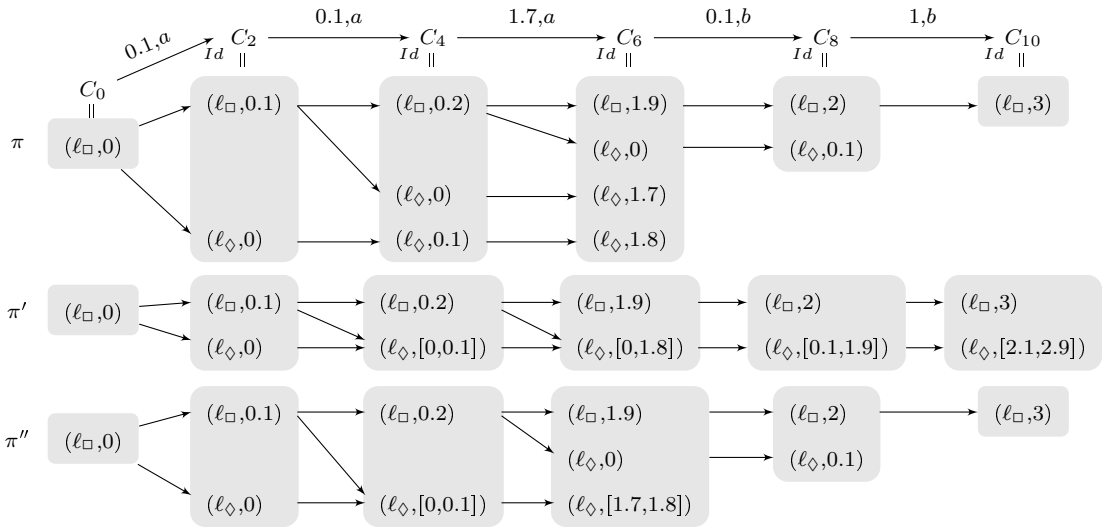


Figure 3.4: Several OCATA runs.

Definition 3.20. *A finite f-run is* accepting *iff its last configuration C_{2n} is accepting and we say that a finite timed word is f-accepted by $\mathcal{A}$ iff there exists an accepting finite f-run of $\mathcal{A}$ on this word.*
We note $L_f(\mathcal{A})$ the language of all finite timed words f-accepted by $\mathcal{A}$.

Nevertheless, to define when an infinite timed word is accepted by $\mathcal{A}$, we need to use the DAG characterisation of f-runs.

Definition 3.21. *We call* branch *of an f-run represented by a DAG G a (finite or) infinite path in G_π. We note $Bran^\omega(G_\pi)$ the set of all* infinite *branches of G_π and, for $\beta \in Bran^\omega(G_\pi)$, we note $Inf(\beta)$ the set of locations occurring infinitely often along β. Such an f-run is* accepting *iff $\forall \beta \in Bran^\omega(G_\pi)$, $Inf(\beta) \cap F \neq \emptyset$ (i.e. we consider Büchi acceptance condition). We say that an infinite timed word θ is f-accepted by $\mathcal{A}$ iff there exists an accepting f-run of $\mathcal{A}$ on θ.*
We note $L_f^\omega(\mathcal{A})$ the language of all infinite timed words f-accepted by $\mathcal{A}$.

Let us observe that the standard semantics for OCATA (where clock valua-

tions are punctual values instead of intervals) is a particular case of the interval semantics, obtained by using the approximation function Id. For the sake of simplicity, we will use the following terms to mention the Id-semantics: we will simply call *run* an Id-run and *accepting run* an Id-accepting run ; we will simply note $\xrightarrow{\sigma}$ instead of $\xrightarrow{\sigma}_{Id}$ and $\xrightarrow{t,\sigma}$ instead of $\xrightarrow{t,\sigma}_{Id}$.

Example 3.22. Let us observe again the three runs of OCATA $\mathcal{A}_\Phi$ (see Figure 3.3) represented in Figure 3.4: they are runs on the finite timed word $\theta = (a, 0.1)(a, 0.2)(a, 1.9)(b, 2)(b, 3)$. π' is not accepting because its last configuration is not accepting: it contains the state $(\ell_\Diamond, [2.1, 2.9])$. In contrary, π and π'' are accepting. Let us call f the approximation function used in π' and g that used in π''. We so have: $\theta \in L_{Id}(\mathcal{A}_\Phi)$ (thanks to π) and $\theta \in L_g(\mathcal{A}_\Phi)$ (thanks to π''). This example does not enable to verify if θ is in $L_f(\mathcal{A}_\Phi)$: π' is not an accepting g-run, but there could exist an *accepting* g-run of $\mathcal{A}_\Phi$ on θ.
These runs are also run prefixes for the infinite timed word

$$\theta' = (a, 0.1)(a, 0.2)(a, 1.9)(b, 2)(b, 3)(b, 4)(b, 5) \dots .$$

It is easy to see that π is an accepting run of $\mathcal{A}_\Phi$ on θ', and that π'' is an accepting g-run of $\mathcal{A}_\Phi$ on θ': this is because the reading of the following b's will only extend the unique branch of each of these runs by configurations containing one state whose location will be $\ell_\Box$. In contrary, π' will not be an accepting f-run of $\mathcal{A}_\Phi$ on θ': it is easy to see that there is no possible way to leave the non-accepting location $\ell_\Diamond$ of $\mathcal{A}_\Phi$ when extending the branch of π' leading to $(\ell_\Diamond, [2.1, 2.9])$. We so have: $\theta \in L^\omega_{Id}(\mathcal{A}_\Phi)$ and $\theta \in L^\omega_g(\mathcal{A}_\Phi)$, but this example does not enable to verify if θ is in $L^\omega_f(\mathcal{A}_\Phi)$.

We close this section by a proposition showing the impact of approximation functions on the accepted language of OCATA: they can only lead to *under-approximations* of $L(\mathcal{A})$ and $L^\omega(\mathcal{A})$. To simplify the proof, we first make some remarks about the form of a minimal model of $\delta(\ell, \sigma)$.

Remark 3.23. Let δ be the transition function of some OCATA, ℓ be a location and σ be a letter. We assume $\delta(\ell, \sigma) = \bigvee_k a_k$, where each a_k is a conjunction

of atoms of the form: ℓ', $x.\ell'$, $x \bowtie c$, $0 \bowtie c$, $\top$ or $\bot$. Then, we observe that *each minimal model* of $\delta(\ell, \sigma)$ with respect to some interval I corresponds to *firing one of the arcs* (ℓ, σ, a_k) *from* (ℓ, I). That is, each minimal model can be obtained by choosing an a_k from $\delta(\ell, \sigma)$, and applying the following procedure. Assume $a_k = \ell_1 \wedge \cdots \wedge \ell_n \wedge x.(\ell_{n+1} \wedge \cdots \wedge \ell_m) \wedge \varphi$, where φ is a conjunction of clock constraints. Then, a_k is firable from a minimal model (ℓ, σ) iff $I \models \varphi$ (otherwise, no minimal model can be obtained from a_k). In this case, the minimal model is $A \cup B^A$, with $A = \{(\ell_i, I) | 1 \leqslant i \leqslant n\}$ and $B^A = \{(\ell_i, [0,0]) \mid n+1 \leqslant i \leqslant m$ and there is no $(\ell_i, I) \in A$ with $0 \in I\}$.

Remark 3.24. Now, let us observe the particular case of an OCATA $\mathcal{A}_\Phi = (\Sigma, L^\Phi, \ell_0^\Phi, F^\Phi, \delta^\Phi)$ for an MTL formula Φ (see Definition 2.164). In its definition, we can see that there is no arc in $\mathcal{A}_\Phi$ containing both ℓ and $x.\ell$ for a certain $\ell \in L^\Phi$. In fact, for each location $\ell \in L^\Phi$, for each $\sigma \in \Sigma$, all *arcs* in δ^Φ are either of the form $(\ell, \sigma, \mathit{true})$, or $(\ell, \sigma, \mathit{false})$, or $\big(\ell, \sigma, \ell \wedge x.(\ell_1 \wedge \cdots \wedge \ell_k) \wedge g\big)$, or of the form $\big(\ell, \sigma, x.(\ell_1 \wedge \cdots \wedge \ell_k) \wedge g\big)$, where g is a guard on x. Hence, the minimal model of $\delta(\ell, \sigma) = \bigvee_k a_k$ firing a_k is simply $\{(\ell_i, I) \mid 1 \leqslant i \leqslant n\} \cup \{(\ell_i, [0,0]) \mid n+1 \leqslant i \leqslant m\}$.
From now on, we consider that $\delta(\ell, \sigma)$ is always written in disjunctive normal form, i.e. $\delta(\ell, \sigma) = \bigvee\limits_{k \in K} a_k$, for certain arcs a_k, atoms as described in the previous remark. Then, when considering an automaton $\mathcal{A}_\Phi$ for an MTL formula Φ, each minimal model M of $\delta(\ell, \sigma)$ with respect to I has the form $a_k[I]$, for some $k \in K$, where

$$a_k[I] = \{(\ell, J) \mid \ell \text{ is a conjunct of } a_k\} \cup \{(\ell, \{0\}) \mid x.\ell \text{ is a conjunct of } a_k\}$$

and I satisfies the guard of a_k.

The following proposition states that $L_f(\mathcal{A}) \subseteq L(\mathcal{A})$ and $L_f^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{A})$. The idea of the proof is the following. Let $C_0 = \{(\ell_0, [0,0])\} \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1}_f C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2}_f \ldots \overset{t_n}{\rightsquigarrow} C_{2n-1} \xrightarrow{\sigma_n}_f C_{2n} \ldots$ be an *accepting* f-run of $\mathcal{A}$ on θ. We will inductively build, an accepting run $D_0 = \{(\ell_0, 0)\} \overset{t_0}{\rightsquigarrow} D_1 \xrightarrow{\sigma_0} D_2 \overset{t_1}{\rightsquigarrow} D_3 \xrightarrow{\sigma_1} \ldots \overset{t_n}{\rightsquigarrow} D_{2n-1} \xrightarrow{\sigma_n} D_{2n} \ldots$ on θ such that the following *invariant* holds:

for all $1 \leqslant i \leqslant 2n$, for all $(\ell, [v, v]) \in D_i$, there is $(\ell, I) \in C_i$ such that $v \in I$.

The basis case is trivial since $C_0 = D_0$. For the inductive case, we first observe that the elapsing of time maintains the invariant. Then, we show that each discrete step in the f-run can be simulated by a discrete step in the run that maintains the invariant. Indeed, if a σ-labelled transition is firable from (ℓ, I), it is also firable from all (ℓ, v) with $v \in I$. Remark that the converse is not true: there might be a set of σ-labelled transitions that are firable from each (ℓ, v) with $v \in I$, but no σ-labelled transition firable from (ℓ, I), because *all clock values* in I must satisfy the guard of a *unique* transition. This is why to use approximation functions only leads to *under-approximations* of $L(\mathcal{A})$ and $L^\omega(\mathcal{A})$.

Proposition 3.25. For all OCATA $\mathcal{A}$, for all $f \in APP_{\mathcal{A}}$: $L_f(\mathcal{A}) \subseteq L(\mathcal{A})$ and $L_f^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{A})$.

Proof. Let $f \in APP_{\mathcal{A}}$ and $\theta = (\overline{\sigma}, \overline{\tau}) \in L_f(\mathcal{A})$ such that $|\theta| \in \{n, +\infty\}$. There is an accepting f-run of $\mathcal{A}$ on θ, say $\pi : C_0 = \{(\ell_0, [0, 0])\} \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_0}_f C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2}_f \ldots \overset{t_i}{\rightsquigarrow} C_{2i-1} \xrightarrow{\sigma_n}_f C_{2i} \ldots$. We are looking for an accepting run π' of $\mathcal{A}$ on θ, it must be on the form $D_0 = \{(\ell_0, 0)\} \overset{t_0}{\rightsquigarrow} D_1 \xrightarrow{\sigma_0} D_2 \overset{t_1}{\rightsquigarrow} D_3 \xrightarrow{\sigma_1} \ldots \overset{t_i}{\rightsquigarrow} D_{2i-1} \xrightarrow{\sigma_i} D_{2i} \ldots$.
We construct π' by induction, the induction hypothesis is the following. We have constructed π' until configuration D_{2j} and the following property is satisfied:

$$\forall (\ell, v) \in D_{2j}, \exists (\ell, I) \in C_{2j} : v \in I \qquad (*j).$$

<u>Basis:</u> $(j = 0)$ We must construct $D_0 := \{(\ell_0, 0)\}$. As $C_0 = D_0$, $(*j)$ is satisfied for $j = 0$.
<u>Induction:</u> $(j = k + 1)$ suppose that for all $j \leqslant k$, we have constructed π' until configuration D_{2j} and the property $(*j)$ is satisfied. We can construct π' until configuration $D_{2.(k+1)}$ and the property $(*k + 1)$ is still satisfied.

- We construct $D_{2k+1} := D_{2k} + t_{k+1}$ what corresponds to the transition $D_{2k} \overset{t_{k+1}}{\rightsquigarrow} D_{2k+1}$ of π'. In π, $C_{2k} \overset{t_{k+1}}{\rightsquigarrow} C_{2k+1}$, and so $C_{2k+1} := C_{2k} + t_{k+1}$. Thanks to induction hypothesis (for $i = k$), it is straightforward to see that:

$$\forall(\ell, v) \in D_{2k+1}, \exists(\ell, I) \in C_{2k+1} : v \in I. \tag{3.1}$$

- We must construct D_{2k+2} corresponding to the transition $D_{2k+1} \xrightarrow{\sigma_{k+1}} D_{2k+2}$. By definition of $\longrightarrow$, we know that, noting $D_{2k+1} = \{(\ell_j, v_j)_{j \in J}\}$, D_{2k+2} must be written as $\bigcup_{l \in L} M_l$, where M_l is a minimal model of $\delta(\ell_l, \sigma_{k+1})$ with respect to v_l. Let $(\ell_l, v_l) \in D_{2k+1}$, we will construct M_l.
  In π, $C_{2k+1} \xrightarrow{\sigma_{k+1}}_f C_{2k+2}$. So, noting $C_{2k+1} = \{(\ell_{l'}, I_{l'})_{l' \in L'}\}$, $C_{2k+2} \in f(E)$, where E is the union (where non-disjoint interval have been managed) of minimal models, say $\mathrm{E}_{l'}$, of $\delta(\ell_{l'}, \sigma_{k+1})$ with respect to $I_{l'}$. Each minimal model corresponds to taking one possible arc $a_{l'}$ starting from $\ell_{l'}$ whose clock constraint is satisfied by $I_{l'}$. Thanks to (3.1), we know that $\exists(\ell_l, I_l) \in C_{2k+1}$ such that $v_l \in I_l$. We can take the arc a_l from (ℓ_l, v_l) because as I_l satisfy its clock constraint, in particular $v_l \in I_l$ satisfies it. This way, we find M_l, a minimal model of $\delta(\ell_l, \sigma_{k+1})$ with respect to v_l. We let $D_{2k+2} = \bigcup_{l \in L} M_l$ and $\delta(\ell_l, \sigma_{k+1}) = \bigvee_{h \in H} a_h$. We know that for all $l \in L$, $M_l = a_h[v_l]$ and $E_l = a_h[I_l]$, for a certain $h \in H$. Thanks to (3.1), we so have:

$$\forall(\ell, v) \in D_{2k+2}, \exists(\ell, I) \in E : v \in I. \tag{3.2}$$

  It remains to prove that $\forall(\ell, v) \in D_{2.(k+1)}, \exists(\ell, I) \in C_{2.(k+1)}$ such that $v \in I$. This follows from (3.2) and the fact that $C_{2(k+1)} \in f(E)$, what ensure in particular that:

$$\forall(\ell, H) \in E, \exists(\ell, I) \in C_{2(k+1)} : H \subseteq I.$$

To conclude that $L_f(\mathcal{A}) \subseteq L(\mathcal{A})$, suppose that $|\theta| = n$. As π is accepting, C_{2n} is accepting and we must show that so is D_{2n}. The previous induction constructs the run π' and shows that, in particular, $\forall(\ell, v) \in D_{2n}, \exists(\ell, I) \in C_{2n}$ such that

$v \in I$ (property $(*n)$). As C_{2n} is accepting, all the states it contains are accepting, i.e. $\forall(\ell, I) \in C_{2n}$, ℓ is an accepting location. We deduce from this that D_{2n} is an accepting configuration, so that π' is an accepting run.
To conclude that $L_f^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{A})$, suppose that $|\theta| = +\infty$ and let us consider an infinite branch $\beta' = ((\ell_0, v_0), 0)((\ell_1, v_1), 1) \dots ((\ell_i, v_i), i) \dots$ of π'. We must show that there are infinitely many $j \geqslant 0$ such that $\ell_j \in F$. Thanks to the previous induction, it is not difficult to see that there is a corresponding infinite branch $\beta = ((\ell_0, I_0), 0)((\ell_1, I_1), 1) \dots ((\ell_i, I_i), i) \dots$ in π, containing the same suite of locations (such that $\forall j \geqslant 0$, $v_j \in I_j$). As π is accepting, there are infinitely many $j \geqslant 0$ such that $\ell_j \in F$, what we wanted to show. $\square$

Our following aim is to find a k-bounded approximation function f such that $L_f(\mathcal{A}) = L(\mathcal{A})$ and $L_f^\omega(\mathcal{A}) = L^\omega(\mathcal{A})$. However, for a general $\mathcal{A}$, these equalities are not verified.
For instance, let us observe the OCATA $\mathcal{A}$ of Figure 3.5. The run π of $\mathcal{A}$ on the timed word $\theta = (a, 0.1)(a, 0.2)(a, 0.9)(a, 1.2)$ is represented in Figure 3.6 (top). If an approximation function merges two clock copies, of values smaller than 1, present in location ℓ_1, this should lead to a 'blocking' situation from which no run could result (see Figure 3.6 (bottom), for instance). Indeed, we could reach a state (ℓ_1, I) for an interval I containing both 1 and a certain $n \neq 1$: no arc can be taken from this state. Now, let us consider a k-bounded approximation function f. We recall that there could be an unbounded number of clock copies associated with location ℓ_1 of values smaller than 1 (see Example 2.101 for details). Hence, because of the blocking situations produced by the merging of clock copies of values smaller than 1 (such groupings are unavoidable for timed words containing more than k a's within 1 time unit), we easily admit that $L_f(\mathcal{A}) \subsetneq L(\mathcal{A})$.
In the following chapter, we will show that, if we consider an MITL formula Φ and the associated OCATA $\mathcal{A}_\Phi$, there always exists a bound k_Φ and a k_Φ-bounded approximation function $f_\Phi^\star$ such that: $L_{f_\Phi^\star}(\mathcal{A}_\Phi) = L(\mathcal{A}_\Phi)$.
In chapter 5, we will establish the twin result in the setting of infinite words: for the same k_Φ-bounded approximation function $f_\Phi^\star$ than used over finite words, we

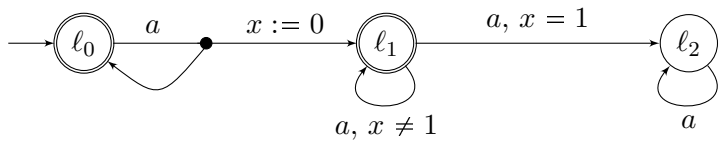


Figure 3.5: OCATA $\mathcal{A}$

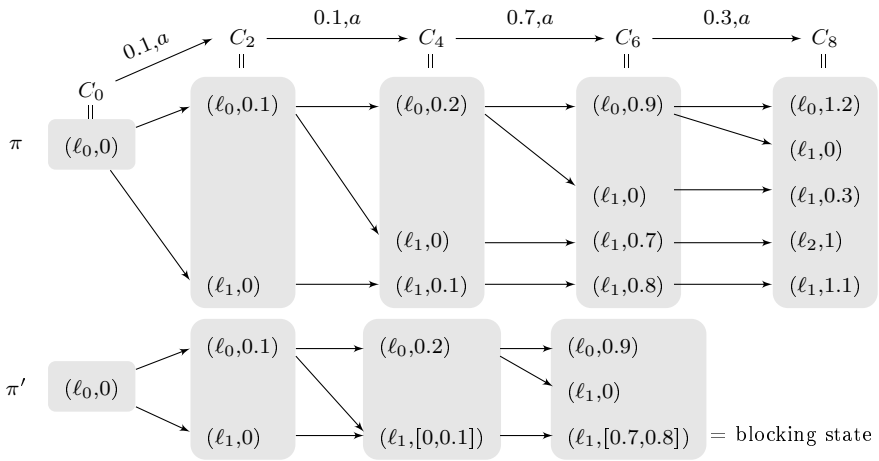


Figure 3.6: Run and f-run of $\mathcal{A}$.

will show that $L^{\omega}_{f^{\star}_{\Phi}}(\mathcal{A}_{\Phi}) = L^{\omega}(\mathcal{A}_{\Phi})$.

CHAPTER 4

MITL satisfiability and model-checking over finite words

In this chapter, we focus on the finite words setting. Thanks to the novel semantics for OCATA presented in the previous chapter, we can now present our new translation, from any MITL formula Φ, to a *timed automaton* that accepts $[\![\Phi]\!]$. This is the object of Section 4.1. The advantage of our construction is that we build this timed automaton from the alternating timed automaton $\mathcal{A}_\Phi$ *in a very intuitive way*, by opposition to the intricate construction presented in [5]. In section 4.2, we display our technique to perform 'on the fly' MITL satisfiability and model-checking over finite words, using directly $\mathcal{A}_\Phi$ (whose size is linear in the size of Φ) and so avoiding, in general, to construct the whole timed automaton accepting $[\![\Phi]\!]$ (whose size is exponential in the size of Φ). These basic algorithms use a region abstraction. We then show, in Section 4.3, how we can use antichains in way to improve those algorithms. Section 4.4 is dedicated to the presentation of other algorithms, based on a zone abstraction, to solve the MITL satisfiability and model-checking problems over finite words. As for the

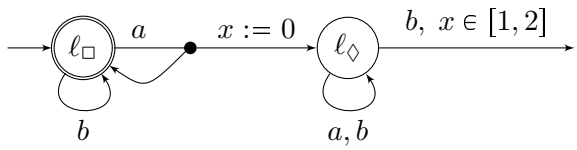


Figure 4.1: OCATA $\mathcal{A}_\Phi$ with $\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,2]} b)$.

region-based algorithm, we show how we can exploit antichains to try to improve the practical results obtained thanks to those algorithms: this is the object of Section 4.5.
The results displayed in this chapter are a part of our publication: [19] (arXiv reference: [20]).

4.1 From MITL to timed automata

In this section, we present our new technique to build, from any MITL formula Φ, a *timed automaton* that accepts $[\![\Phi]\!]$, i.e. the set of timed words satisfying Φ. We proceeds starting from the OCATA $\mathcal{A}_\Phi$ (see Section 2.2). However, in general, it is not possible to translate an OCATA into a timed automaton recognizing the same language. Indeed, all along its runs, an OCATA creates clock copies. In general, the number of clock copies created this way is *unbounded* and it is not possible to simulate them by a timed automaton with finitely many clocks. Our technique then relies on the definition of a family of *bounded approximation functions* $f^\star_\Phi$, such that, for all MITL formula Φ, $L^\omega_{f^\star_\Phi}(\mathcal{A}_\Phi) = L^\omega(\mathcal{A}_\Phi)$. Since each $f^\star_\Phi$ is a *bounded approximation function*, the number of clock copies in the $f^\star_\Phi$-semantics of $\mathcal{A}_\Phi$ is *bounded*. This allows us to build a *timed automaton* $\mathcal{B}_\Phi$ *accepting* $[\![\Phi]\!]$.

Example 4.1. Let us illustrate the idea behind the approximation function $f^\star_\Phi$, for $\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,2]} b)$, by considering the run prefixes on

$$\theta = (a, 0.1)(a, 0.2)(a, 1.9)(b, 2)(b, 3)(b, 4)\ldots$$

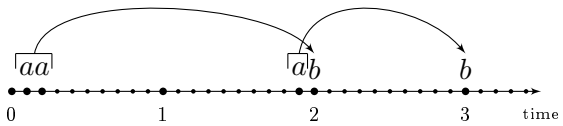


Figure 4.2: The grouping of clocks.

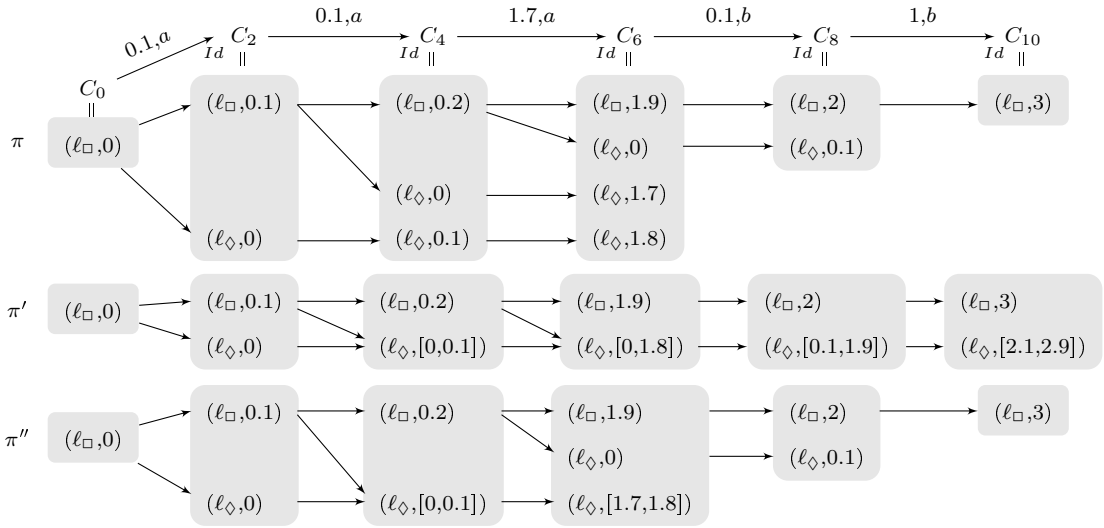


Figure 4.3: Several OCATA runs.

in Figure 4.3. The two first positions (with $\sigma_1 = \sigma_2 = a$) of θ satisfy $\Diamond_{[1,2]}b$, *thanks to the* b *in position* 4 (with $\tau_4 = 2$), while position 3 (with $\sigma_3 = a$) satisfies $\Diamond_{[1,2]}b$ *thanks to the* b *in position* 5 (with $\tau_5 = 3$), see Figure 4.2. Hence, $f^\star_\Phi$ groups the two clock copies created in $\ell_\Diamond$ when reading the two first a's, but keeps the third one apart. This yields the $f^\star_\Phi$-run π'' in Figure 4.3. On the other hand, the strategy of grouping all the clock copies present in each location, which yields π', is not a good solution. This prefix cannot be extended to an accepting run because of the copy in state $(\ell_\Diamond, [2.1, 2.9])$ in the rightmost configuration, that will never be able to visit an accepting location.

4.1.1 The approximation functions $f^{\star}_{\Phi}$

Let us now formally define the family of *bounded* approximation functions that will form the basis of our translation from MITL to timed automata.

Throughout this description, we assume an OCATA $\mathcal{A}$ with set of locations L. Let $S = \{(\ell, I_0), (\ell, I_1), \ldots, (\ell, I_m)\}$ be a set of states of $\mathcal{A}$, all in the same location ℓ, with $I_0 < I_1 < \cdots < I_m$. Then, we let

$$\begin{array}{lll} \mathsf{Merge}\,(S) & = \{(\ell, [0, sup(I_1)]), (\ell, I_2), \ldots, (\ell, I_m)\} & \text{if } I_0 = [0,0] \text{ and} \\ \mathsf{Merge}\,(S) & = S & \text{otherwise,} \end{array}$$

i.e. $\mathsf{Merge}\,(S)$ is obtained from S by *grouping* I_0 and I_1 iff $I_0 = [0,0]$, otherwise $\mathsf{Merge}\,(S)$ does not modify S. Observe that, in the former case, if I_1 is not a singleton, then $\|\mathsf{Merge}\,(S)\| = \|S\| - 1$. Now, we can lift the definition of Merge to configurations, with the addition of a bound k representing the maximal size we want our configurations to have. The function Merge defines a simple manner to group the clock copies associated with a same location. We will see that, grouping the clock copies this way is sufficient to obtain a bound on their number.

Definition 4.2. *Let C be a configuration of $\mathcal{A}$ and let $k \in \mathbb{N}$. We let:*

$$\mathsf{Merge}\,(C, k) = \left\{C' \mid \|C'\| \leqslant k \text{ and } \forall \ell \in L : C'(\ell) \in \{\mathsf{Merge}\,(C(\ell))\,, C(\ell)\}\right\}$$

Observe that $\mathsf{Merge}\,(C, k)$ is a (possibly empty) *set of configurations*, where each configuration (i) has at most k clock copies, and (ii) can be obtained by applying or not the Merge function to each $C(\ell)$.

Example 4.3. Let us consider the OCATA $\mathcal{A}$ of Figure 4.1 and its configuration $C = \{(\ell_\Box, [7.6, 7.6]), (\ell_\Diamond, [0,0]), (\ell_\Diamond, [0.3, 0.7]), (\ell_\Diamond, [0.8, 0.9])\}$. We have: $C(\ell_\Box) = \{(\ell_\Box, [7.6, 7.6])\}$ and $C(\ell_\Diamond) = \{(\ell_\Diamond, [0,0]), (\ell_\Diamond, [0.3, 0.7]), (\ell_\Diamond, [0.8, 0.9])\}$. $\mathsf{Merge}\,(C(\ell_\Box)) = \{(\ell_\Box, [7.6, 7.6])\}$ and $\mathsf{Merge}\,(C(\ell_\Diamond)) = \{(\ell_\Diamond, [0, 0.7]), (\ell_\Diamond, [0.8, 0.9])\}$. If we want to bound the number of clock copies at 6, we can use $\mathsf{Merge}\,(C, 6) = \{\{(\ell_\Box, [7.6, 7.6]), (\ell_\Diamond, [0, 0.7]), (\ell_\Diamond, [0.8, 0.9])\}\}$. If we admit a bound of 8 clock

copies, we can use $\mathsf{Merge}\,(C,8) = \{\{(\ell_\Box,[7.6,7.6]),(\ell_\Diamond,[0,0.7]),(\ell_\Diamond,[0.8,0.9])\}, \{(\ell_\Box,[7.6,7.6]),(\ell_\Diamond,[0,0]),(\ell_\Diamond,[0.3,0.7]),(\ell_\Diamond,[0.8,0.9])\}\}$.

Let us now define a family of k-bounded approximation functions, based on Merge.

Definition 4.4. *Let $k \geqslant 2|L|$ be a bound and let C be a configuration, assuming that $C(\ell) = \{I_1^\ell, \ldots, I_{m_\ell}^\ell\}$ for all $\ell \in L$. Then:*

$$F^k(C) = \begin{cases} \mathsf{Merge}\,(C,k) & \textit{if } \mathsf{Merge}\,(C,k) \neq \varnothing \\ \big\{(\ell,[inf(I_1^\ell), sup(I_{m_\ell}^\ell)]) \mid \ell \in L\big\} & \textit{otherwise.} \end{cases}$$

Roughly speaking, the $F^k(C)$ function tries to obtain configurations C' that approximate C and such that $\|C'\| \leqslant k$, using the Merge function. If it fails to, i.e., when $\mathsf{Merge}\,(C,k) = \varnothing$, $F^k(C)$ returns a single configuration, obtained from C by grouping all the intervals in each location. The latter case only occurs in the definition of F^k for the sake of completeness.

When the OCATA $\mathcal{A}$ has been obtained from an MITL formula Φ, and for k big enough (see hereunder) each $\theta \in [\![\Phi]\!]$ will be recognised by at least one F^k-run of $\mathcal{A}$ that traverses only configurations obtained thanks to Merge. We can now finally define $f_\Phi^\star$ for every MITL formula Φ. This definition is based on a bound $M(\Phi)$ which will be given by the proof of Theorem 4.7. Its (quite intricate) value will be given later, in Definition 4.10.

Definition 4.5. *Let Φ be an MITL formula. We let $f_\Phi^\star = F^K$, where $K = \max\{2|L|, M(\Phi)\}$ and $M(\Phi)$ is a bound given by Theorem 4.7 and formally defined in Definition 4.10.*

Example 4.6. Let us observe the OCATA $\mathcal{A}$ of Figure 4.4, corresponding to the MITL formula $\Phi \equiv \Box\big(a \Rightarrow \big(\Diamond_{[0,1]} b \wedge \Diamond_{[0,1]} c\big)\big)$. It is an OCATA on the alphabet $\Sigma = \{a,b,c\}$ and its set of locations is $L = \{\ell_\Box, \ell_{\Diamond b}, \ell_{\Diamond c}\}$. Reading the beginning of timed word $\theta = (a,0)(a,0.1)(a,0.2)(a,0.5)$, we may reach the following configuration:

$$C = \{(\ell_\Box,0.5),(\ell_{\Diamond b},0),(\ell_{\Diamond b},0.3),(\ell_{\Diamond b},[0.4,0.5]),(\ell_{\Diamond c},0),(\ell_{\Diamond c},0.3),(\ell_{\Diamond c},[0.4,0.5])\}.$$

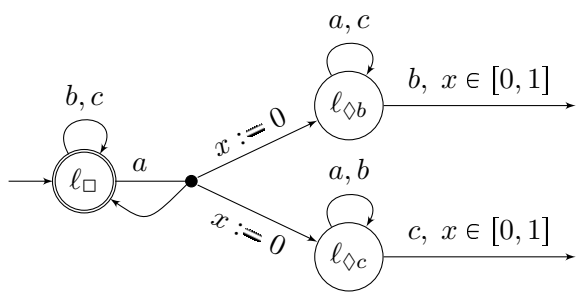


Figure 4.4: An OCATA $\mathcal{A}$ for formula $\Phi \equiv \Box\big(a \Rightarrow \big(\Diamond_{[0,1]} b \wedge \Diamond_{[0,1]} c\big)\big)$.

Let us consider a bound $k = 6$ ($\geqslant 2|L|$) on the number of clock copies. Observe that $\mathsf{Merge}\,(C(\ell_\Box)) = \{(\ell_\Box, 0.5)\}$, $\mathsf{Merge}\,(C(\ell_{\Diamond b})) = \{(\ell_{\Diamond b}, [0, 0.3]), (\ell_{\Diamond b}, [0.4, 0.5])\}$ and $\mathsf{Merge}\,(C(\ell_{\Diamond c})) = \{(\ell_{\Diamond c}, [0, 0.3]), (\ell_{\Diamond c}, [0.4, 0.5])\}$. So, $\|\mathsf{Merge}\,(C(\ell_\Box))\| = 1$, $\|\mathsf{Merge}\,(C(\ell_{\Diamond b}))\| = 4$ and $\|\mathsf{Merge}\,(C(\ell_{\Diamond c}))\| = 4$; and hence $\mathsf{Merge}\,(C, 6) = \varnothing$. In this case, $F^6(C) = \{(\ell_\Box, 0.5), (\ell_{\Diamond b}, [0, 0.5]), (\ell_{\Diamond c}, [0, 0.5])\}$. When $M(\Phi)$ will be formally defined, we will see that $M(\Phi) = 5$, so that $f^\star_\Phi = F^6$. The intuition is that this case $\mathsf{Merge}\,(C, 6) = \varnothing$ was reached because, previously in the run, 'wrong' grouping have been done. Indeed, the proof of Theorem 4.7 gives a criterion characterizing the situations in which clock copies should be grouped. If the groupings are made as stipulated by this proof, the case $\mathsf{Merge}\,(C, 6) = \varnothing$ is never reached.

For an MITL formula Φ, it is easy to see that $f^\star_\Phi$ is indeed a *bounded approximation function*. Then, we can show the main theorem of this section, which says that, for all MITL formula Φ, the $f^\star_\Phi$-semantics of $\mathcal{A}_\Phi$ accepts exactly $[\![\Phi]\!]$:

Theorem 4.7. *For all MITL formula Φ, $f^\star_\Phi$ is a* bounded approximation function *and $L_{f^\star_\Phi}(\mathcal{A}_\Phi) = L(\mathcal{A}_\Phi) = [\![\Phi]\!]$.*

To simplify the following proofs, we deviate slightly from the definition of $\mathcal{A}_\Phi$ (Definition 2.164), and assume that if a formula of type $\Phi_1 U_I \Phi_2$ or $\Phi_1 \tilde{U}_I \Phi_2$ appears more than once as a sub-formula of Φ, the occurrences of this formula are supposed different and are encoded as different locations.

To prove Theorem 4.7, we rely on two propositions, concerning respectively the U and $\tilde{U}$ operators. The properties given by Propositions 4.8 and 4.9 are crucial to determine, given an accepting run, whether we can *group* several intervals and retain an accepting run or not. This observation will be central in the proof of our main theorem.

To simplify their statement and the proof of Theorem 4.7, we will use the following notations. Assume we observe a timed word $\theta = (\overline{\sigma}, \overline{\tau})$, where $\overline{\sigma} = \sigma_1\sigma_2\ldots\sigma_n$ and $\overline{\tau} = \tau_1\tau_2\ldots\tau_n$. For all $0 \leqslant k \leqslant n$, we denote by $\theta^k = (\overline{\sigma}_k, \overline{\tau}_k)$, where $\overline{\sigma}_k = \sigma_k\sigma_{k+1}\ldots\sigma_n$ and $\overline{\tau}_k = \tau'_1\tau'_2\ldots\tau'_{n-k}$ the infinite timed word such that $\forall 1 \leqslant i \leqslant n-k$, $\tau'_i = \tau_{i+k} - \tau_k$.

Let us also note:

$$\mho(k, i, I, J) :=$$
$$(\theta^k, i) \models \Phi_2 \wedge \tau_i^k \in I - \inf(J) \wedge \tau_i^k \in I - \sup(J) \wedge \forall 1 \leqslant m' < i : (\theta^k, m') \models \Phi_1.$$

Proposition 4.8. Let Φ be an MITL formula, let K be a set of indices and, $\forall k \in K$, let $\Phi_k = \Phi_{1,k} U_{I_k} \Phi_{2,k}$ be subformulas of Φ. For all $k \in K$, let ℓ_{Φ_k} be their associated locations in $\mathcal{A}_\Phi$. Let $\theta = (\overline{\sigma}, \overline{\tau})$ be a timed word and let $J_k \in \mathcal{I}(\mathbb{R}^+)$ be closed intervals.

$$\text{The automaton } \mathcal{A}_\Phi \text{ accepts } \theta \text{ from configuration } \{(\ell_{\Phi_k}, J_k)_{k\in K}\}$$
$$\text{iff}$$
$$\forall k \in K, \exists m_k \geqslant 1 : \mho(0, m_k, I, J).$$

Proposition 4.9. Let Φ be an MITL formula, let K be a set of indices and, $\forall k \in K$, let $\Phi_k := \Phi_{1,k} \tilde{U}_{I_k} \Phi_{2,k}$ be sub-formulas of Φ. For all $k \in K$, let ℓ_{Φ_k} be their associated locations in $\mathcal{A}_\Phi$. Let $\theta = (\overline{\sigma}, \overline{\tau})$ be a timed word and $J_k \in \mathcal{I}(\mathbb{R}^+)$.

$$\text{The automaton } \mathcal{A}_\Phi \text{ accepts } \theta \text{ from configuration } \{(\ell_{\Phi_k}, J_k)_{k\in K}\}$$
$$\text{iff}$$
$$\forall k \in K, \forall v \in J_k, \text{ the automaton } \mathcal{A}_\Psi \text{ accepts } \theta \text{ from configuration } \{(\ell_{\Phi_k}, [v, v])\}$$
$$(\text{i.e.: } \forall k \in K, \forall v \in J_k, (\theta, 1) \models \Phi_{1,k} \tilde{U}_{I_k - v_k} \Phi_{2,k}).$$

The proofs of these two Propositions can be found in the appendix. They are identical in the settings of finite and infinite words.

We can now explain the proof of Theorem 4.7. The definition of $f^\star_\Phi$ guarantees it is a K-bounded approximation function, for $K = \max\{2|L|, M(\Phi)\}$. The equality $L(\mathcal{A}_\Phi) = [\![\Phi]\!]$ have already been established ([51]) and Theorem 3.25 proves the inclusion $L_{f^\star_\Phi}(\mathcal{A}_\Phi) \subseteq L(\mathcal{A}_\Phi)$. Hence, the proof of Theorem 4.7 consists in establishing the last needed inclusion: $L_{f^\star_\Phi}(\mathcal{A}_\Phi) \supseteq L(\mathcal{A}_\Phi)$. The process of the proof is the following: from an accepting run of $\mathcal{A}_\Phi$ on a given timed word, we construct inductively an accepting $f^\star_\Phi$-run: step $k+1$ constructs the $(2k+1)$th and $(2(k+1))$th configurations of this $f^\star_\Phi$-run. In a same time, we prove that the number of clock copies associated with each location of $\mathcal{A}_\Phi$ stays bounded in the $(2k+1)$th and $(2(k+1))$th configurations of this $f^\star_\Phi$-run. This enables to deduce the exact value of the bound $M(\Phi)$ on the number of clock copies needed.

Proof of Theorem 4.7. Let $\theta = (\overline{\sigma}, \overline{\tau}) \in L(\mathcal{A}_\Phi)$, where $\overline{\sigma} = \sigma_1\sigma_2\ldots\sigma_n$ and $\overline{\tau} = \tau_1\tau_2\ldots\tau_n$. There is an accepting run π of $\mathcal{A}_\Phi$ on θ, say $C_0 = \{(\ell_0, 0)\} \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \ldots \overset{t_n}{\rightsquigarrow} C_{2n-1} \xrightarrow{\sigma_n} C_{2n}$. We must find an accepting $f^\star_\Phi$-run π' of $\mathcal{A}_\Phi$ on θ, for a certain bound $M(\Phi)$ on the number of clock copies. Our proof consists in constructing π' grouping clock copies in a unique interval when the criteria of Propositions 4.8 and 4.9 are respected, so that we still have an accepting run from the interval formed. Our method is to group the last clock copy associated with a location ℓ_i with the previous interval associated with this location (as done with the Merge() function) if Propositions 4.8 and 4.9 ensure there is still an accepting run from the formed interval. π' will be denoted $D_0 = \{(\ell_0, J)\} \overset{t_1}{\rightsquigarrow} D_1 \xrightarrow{\sigma_1}_{f^\star_\Phi} D_2 \overset{t_2}{\rightsquigarrow} D_3 \xrightarrow{\sigma_2}_{f^\star_\Phi} \ldots \overset{t_n}{\rightsquigarrow} D_{2n-1} \xrightarrow{\sigma_n}_{f^\star_\Phi} D_{2n}$. We will construct it inductively: at step $k+1$ we construct D_{2k+1} and $D_{2(k+1)}$ and prove there is an accepting run $D_0 \overset{t_1}{\rightsquigarrow} D_1 \xrightarrow{\sigma_1}_{f^\star_\Phi} D_2 \overset{t_2}{\rightsquigarrow} D_3 \xrightarrow{\sigma_2}_{f^\star_\Phi} \ldots \overset{t_{k+1}}{\rightsquigarrow} D_{2k+1} \xrightarrow{\sigma_{k+1}}_{f^\star_\Phi} D_{2(k+1)} \overset{t_{k+2}}{\rightsquigarrow} E^k_{2k+3} \xrightarrow{\sigma_{k+2}}_{f^\star_\Phi} E^k_{2(k+2)} \ldots \overset{t_n}{\rightsquigarrow} E^k_{2n-1} \xrightarrow{\sigma_n}_{f^\star_\Phi} E^k_{2n}$, thanks to Propositions 4.8 and 4.9 ; at step $k+2$, we construct D_{2k+3} and

$D_{2(k+2)}$ from E^k_{2k+3} and $E^k_{2(k+2)}$.[1] In the same time, we will prove that, in each configuration $D = \bigcup_{\ell \in L} D(\ell)$ reached by π', if ℓ corresponds to a formula $\varphi_U \equiv \Phi_1 U_I \Phi_2$, $\|D(\ell_i)\| \leqslant 4.\lceil \frac{\inf(I)}{|I|} \rceil + 2$; and, if ℓ_i corresponds to a formula $\varphi_{\tilde{U}} \equiv \Phi_1 \tilde{U}_I \Phi_2$, $\|D(\ell)\| \leqslant 2.\lceil \frac{\sup(I)}{|I|} \rceil + 2$. These properties on the number of clock copies will enable us to deduce the value of $M(\Phi)$ needed in the definition of $f^\star_\Phi$. In the sequel, we denote $M(\varphi_{U,I}) := 4.\lceil \frac{\inf(I)}{|I|} \rceil + 2$ and $M(\varphi_{\tilde{U},I}) := 2.\lceil \frac{\sup(I)}{|I|} \rceil + 2$.

The induction hypothesis (at step k+1) is that we have the following *accepting* run on θ:

$$D_0 \overset{t_1}{\rightsquigarrow} D_1 \cdots D_{2k-1} \xrightarrow{\sigma_k}_{f^\star_\Phi} D_{2k} \overset{t_{k+1}}{\rightsquigarrow} E^k_{2k+1} \xrightarrow{\sigma_{k+1}} E^k_{2(k+1)} \cdots E^k_{2n-1} \xrightarrow{\sigma_n} E^k_{2n}.$$

It is not yet an $f^\star_\Phi$-run because we have no bound on the number of clock copies for configurations E^k_i, for $2k+1 \leqslant i \leqslant 2n$. Its beginning, until configuration D_{2k}, is an $f^\star_\Phi$-run while its end is a simple run. It is such that $\forall 0 \leqslant j \leqslant 2k, \forall \ell \in L$ corresponding to a formula $\Phi_1 U_I \Phi_2$, we have that $\|D_j(\ell)\| \leqslant M(\varphi_{U,I})$ and $\forall \ell \in L$ corresponding to a formula $\Phi_1 \tilde{U}_I \Phi_2$, we have that $\|D_j(\ell)\| \leqslant M(\varphi_{\tilde{U},I})$. Thanks to this hypothesis, we will show how to build D_{2k+1} and D_{2k+2} in way:

(i) $\pi^\star \equiv D_0 \overset{t_1}{\rightsquigarrow} D_1 \xrightarrow{\sigma_1} D_2 \cdots D_{2k} \overset{t_{k+1}}{\rightsquigarrow} D_{2k+1} \xrightarrow{\sigma_{k+1}} D_{2k+2}$ is the beginning of an accepting $f^\star_\Phi$-run of $\mathcal{A}_\Phi$ on θ ;

(ii) $D_{2k+2} \overset{t_{k+2}}{\rightsquigarrow} E^{k+1}_{2k+3} \cdots E^{k+1}_{2n-1} \xrightarrow{\sigma_n} E^{k+1}_{2n}$ is an accepting end of run, for the beginning of $f^\star_\Phi$-run $\pi^\star$ of $\mathcal{A}_\Phi$ on θ ;

(ii) $\forall \ell \in L$ corresponding to a formula $\Phi_1 U_I \Phi_2$: $\|D_{2k+2}(\ell)\| \leqslant M(\varphi_{U,I})$, and $\forall \ell_i \in L$ corresponding to a formula $\Phi_1 \tilde{U}_I \Phi_2$: $\|D_{2k+2}(\ell)\| \leqslant M(\varphi_{\tilde{U},I})$.

<u>Basis:</u> (k=0) we define $D_0 = C_0 = \{(\ell_0, [0,0])\}$. We still have an accepting run of $\mathcal{A}_\Phi$ on θ: π.

[1] To construct D_{2k+3} and $D_{2(k+2)}$ from E^k_{2k+3} and $E^k_{2(k+2)}$ enables to simplify the proof and to simply define $f^\star_\Phi$, although they could be directly constructed from C_{2k+3} and $C_{2(k+2)}$

Induction: We know there is an accepting run of $\mathcal{A}_\Phi$ from D_{2k} on θ^{k+1} and observe the two first steps of this run: $D_{2k} \overset{t_{k+1}}{\rightsquigarrow} E^k_{2k+1} \xrightarrow{\sigma_{k+1}} E^k_{2(k+1)}$. We define configuration D_{2k+1} of π' as $D_{2k+1} := E^k_{2k+1}$. As $\forall \ell \in L$ corresponding to a formula $\Phi_1 U_I \Phi_2$, $\|D_{2k}(\ell)\| \leqslant M(\varphi_{U,I})$ (induction hypothesis), we also have $\|D_{2k+1}(\ell)\| \leqslant M(\varphi_{U,I})$. In the same way, as $\forall \ell \in L$ corresponding to a formula $\Phi_1 \tilde{U}_I \Phi_2$, $\|D_{2k}(\ell)\| \leqslant M(\varphi_{\tilde{U},I})$ (induction hypothesis), we also have $\|D_{2k+1}(\ell)\| \leqslant M(\varphi_{\tilde{U},I})$. Assume that $E^k_{2k+2}(\ell) = \{J_1, ..., J_m\}$. We define D_{2k+2} of π' as $f^\star_\Phi(E^k_{2k+2}) = \bigcup_{\ell \in L} f^\star_\Phi(E^k_{2k+2}(\ell))$, where $\forall \ell \in L, f^\star_\Phi(E^k_{2k+2}(\ell))$ is defined as follows:

$$f^\star_\Phi(E^k_{2k+2}(\ell)) = \begin{cases} \mathsf{Merge}\left(E^k_{2k+2}(\ell)\right) & \text{if } \ell \text{ is of type } \Phi_1 U_I \Phi_2 \text{ and} \\ & \quad \exists m \geqslant 1 \text{ such that: } \mho(k+2, m, I, J_2) \\ & \quad \text{or } \ell \text{ is of type } \Phi_1 \tilde{U}_I \Phi_2 \text{ and} \\ & \quad \forall v \in [0, sup(J_2)], (\theta, 1) \models \Phi_1 \tilde{U}_{I-v} \Phi_2 \\ E^k_{2k+2}(\ell) & \text{otherwise} \end{cases}$$

with, as defined above,

$$\mathsf{Merge}\left(E^k_{2k+2}(\ell)\right) = \{(\ell, [0, \sup(J_2)]), (\ell, J_3), (\ell, J_4), \ldots, (\ell, J_n)\}.$$

We must prove there is an accepting run of $\mathcal{A}_\Phi$ from $D_{2.(k+1)} = f^\star_\Phi(E^k_{2k+2}) = \bigcup_{\ell \in L} f(E^k_{2k+2}(\ell_i))$ on θ^{k+2}. Let $\ell \in L$, we will prove that there is an accepting run of $\mathcal{A}_\Phi$ on θ^{k+2} from $D_{2.(k+1)}(\ell) := f^\star_\Phi(E^k_{2k+2}(\ell))$ (which is sufficient). If $f^\star_\Phi(E^k_{2k+2}(\ell)) = E^k_{2k+2}(\ell)$, the accepting run given by induction hypothesis on $E^k_{2k+2}(\ell)$ can always be used. Else,

$$f^\star_\Phi(E^k_{2k+2}(\ell)) = \{(\ell, [0, \sup(J_2)]), (\ell, J_3), (\ell, J_4), \ldots, (\ell, J_n)\}$$

As in this case $E^k_{2k+2}(\ell)$ was $\{(\ell, [0, 0]), (\ell, J_2), (\ell, J_3), (\ell, J_4), \ldots, (\ell, J_n)\}$, the accepting run given by induction hypothesis can be used from $\{(\ell, J_3),\ (\ell, J_4), \ldots, (\ell, J_n)\} \subseteq f^\star_\Phi(E^k_{2k+2}(\ell))$ and we only need to prove there is an accepting run of $\mathcal{A}_\Phi$ on θ^{k+2} from $\{(\ell, [0, \sup(J_2)]\}$.

On the one hand, suppose that ℓ is of type $\Phi_1 U_I \Phi_2$, in this case $\exists m \geqslant 1$ such that $\mho(k+2, m, I, J_2)$ is satisfied: we can conclude thanks to Proposition 4.8. On the other hand, suppose that ℓ is of type $\Phi_1 \tilde{U}_I \Phi_2$, in this case

$\forall v \in [0, sup(J_2)], (\theta, 1) \models \Phi_1 \tilde{U}_{I-v} \Phi_2$, and we can conclude thanks to Proposition 4.9.

We must now show that, the way we grouped clock copies with $f^\star_\Phi$, $\forall \ell \in L$ corresponding to a formula $\Phi_1 U_I \Phi_2$, $\|D_{2k+2}(\ell)\| \leqslant M(\varphi_{U,I})$ and $\forall \ell \in L$ corresponding to a formula $\Phi_1 \tilde{U}_I \Phi_2$, $\|D_{2k+2}(\ell)\| \leqslant M(\varphi_{\tilde{U},I})$. We prove it by contradiction.

For the U case, let us suppose that $\|D_{2k+2}(\ell)\| > M(\varphi_{U,I})$, for a certain location ℓ corresponding to formula $\Phi_1 U_I \Phi_2$. We so have more than $2.\lceil \frac{\inf(I)}{|I|} \rceil + 1$ intervals associated with ℓ in D_{2k+2}, i.e: $D_{2k+2}(\ell) = \{(\ell, J_1), (\ell, J_2), \ldots, (\ell, J_n)\}$, for a certain $n > 2.\lceil \frac{\inf(I)}{|I|} \rceil + 1$. The way we grouped clock copies with $f^\star_\Phi$, we know that each interval J_j, for $1 \leqslant j \leqslant n$, satisfies the following property:

$$\exists k_j \geqslant 1 \text{ such that: } \mho(k+1, k_j, I, J_j). \tag{4.1}$$

Indeed, if it is not the case anymore, π' would not be an accepting run. Moreover, $\forall 1 < j \leqslant n$, we have the following property:

$$\begin{aligned} \forall k_m \geqslant 1 : (\theta^{k+1}, k_m) \not\models \Phi_2 \ \vee \ \tau^{k+1}_{k_m} \notin I - \sup(J_j) \ \vee \ \tau^{k+1}_{k_m} \notin I - \sup(J_{j-1}) \\ \vee \ \exists 1 \leqslant k' < k_m, (\theta^{k+1}, k') \not\models \Phi_1. \end{aligned} \tag{4.2}$$

Indeed, if it is not the case, $\sup(J_{j-1})$ would have been grouped with J_j.

We first claim that $\forall 3 \leqslant j \leqslant n$, $\sup(J_j) - \sup(J_{j-2}) \geqslant |I|$. We will prove it by contradiction. Let $j^\star$ be such that $3 \leqslant j^\star \leqslant n$ and suppose that $\sup(J_{j^\star}) -$

$\sup(J_{j^\star-2}) < |I|$, i.e.: $\sup(J_{j^\star}) < |I| + \sup(J_{j^\star-2})$.

Then: $(I - \sup(J_{j^\star})) \cap (I - \sup(J_{j^\star-2})) \neq \emptyset$

(because these intervals have the same size).

Moreover, $\inf(I - \sup(J_{j^\star}))) < \inf((I - \sup(J_{j^\star-2}))$

(because $\sup(J_{j^\star}) > \sup(J_{j^\star-2})$),

Finally:

$$\begin{aligned}\sup(I - \sup(J_{j^\star})) &= \sup(I) - \sup(J_{j^\star})\\ &> \sup(I) - (|I| + \sup(J_{j^\star-2}))\\ &= \sup(I) - \sup(I) + \inf(I) - \sup(J_{j^\star-2})\\ &= \inf(I) - \sup(J_{j^\star-2})\\ &= \inf(I - \sup(J_{j^\star-2})).\end{aligned}$$

So, as $\sup(J_{j^\star-2}) < \sup(J_{j^\star-1}) < \sup(J_{j^\star})$:

$$I - \sup(J_{j^\star-1}) \subseteq (I - \sup(J_{j^\star})) \cup (I - \sup(J_{j^\star-2})).$$

Considering (4.1) with $j = j^\star$, we have that $\tau'_{k_{j^\star-1}} \in I - \sup(J_{j^\star-1})$, and so

$$\tau'_{k_{j^\star-1}} \in (I - \sup(J_{j^\star})) \cup (I - \sup(J_{j^\star-2})).$$

Though, if $\tau'_{k_{j^\star-1}} \in (I - \sup(J_{j^\star}))$, we contradict (4.2) for $j = j^\star$ taking $k_m = k_{j^\star-1}$ (for the $k_{j^\star-1}$ given by (4.1)) ; and if $\tau'_{k_{j^\star-1}} \in (I - \sup(J_{j^\star-2}))$, we contradict (4.2) for $j = j^\star - 1$ taking again $k_m = k_{j^\star-1}$.

We now know that $\forall 3 \leqslant j \leqslant n$, $\sup(J_j) - \sup(J_{j-2}) \geqslant |I|$. So,

$$\sup(J_n) - \sup(J_1) \geqslant \lceil \frac{(n-2)}{2} \rceil . |I|.$$

As $n > 2.\lceil \frac{\sup(I)}{|I|} \rceil + 1$, we have that:

$$\begin{aligned}\sup(J_n) - \sup(J_1) &> \left\lceil \frac{(2.\left\lceil \frac{\sup(I)}{|I|} \right\rceil + 1 - 2)}{2} \right\rceil .|I|\\ &= \left\lceil \left\lceil \frac{\sup(I)}{|I|} \right\rceil - \frac{1}{2} \right\rceil .|I| = \left\lceil \frac{\sup(I)}{|I|} \right\rceil .|I| \quad \geqslant \quad \sup(I).\end{aligned}$$

It means that $\sup(J_n) - \sup(J_1) > \sup(I)$, and hence $\sup(J_n) > \sup(I)$. It is a contradiction because if $\sup(J_n) > \sup(I)$, we cannot have an accepting run from $\{(\ell, \sup(J_n))\}$ and therefore neither from D_{2k+2}, while we have just proved it is the case.

For the $\tilde{U}$ case, let us suppose that $\|D_{2k+2}(\ell)\| > M(\varphi_{\tilde{U},I})$, for a certain location ℓ corresponding to formula $\Phi_1 \tilde{U}_I \Phi_2$. We so have more than $\lceil \frac{\inf(I)}{|I|} \rceil + 1$ intervals associated with ℓ in D_{2k+2}, i.e.: $D_{2k+2}(\ell) = \{(\ell, J_1), (\ell, J_2), \ldots, (\ell, J_n)\}$, for a certain $n > \lceil \frac{\inf(I)}{|I|} \rceil + 1$. The way we grouped clock copies by intervals, we know that each interval J_j, for $1 \leqslant j \leqslant n$, satisfies the following property:

$$\forall v \in J_j, (\theta, 1) \models \Phi_1 \tilde{U}_{I-v} \Phi_2. \tag{4.3}$$

Indeed, if it is not the case anymore, π' could not be an accepting run. Moreover, $\forall 1 < j \leqslant n$, we have the following property:

$$\exists v \in [\sup(J_{j-1}), \inf(J_j)], (\theta, 1) \not\models \Phi_1 \tilde{U}_{I-v} \Phi_2. \tag{4.4}$$

Indeed, if it is not the case, $\sup(J_{j-1})$ would have been grouped with J_j.)
We first claim that $\forall 2 \leqslant j \leqslant n$, $\inf(J_j) - \inf(J_{j-1}) \geqslant |I|$. We will prove it by contradiction. Let $2 \leqslant j \leqslant n$ and suppose that $\inf(J_j) - \inf(J_{j-1}) < |I|$, i.e.: $\inf(J_j) < |I| + \inf(J_{j-1})$.

Then: $(I - J_j) \cap (I - J_{j-1}) \neq \varnothing$, because:

$$\begin{aligned}
\sup(I - J_j) = \sup(I) - \inf(J_j) &> \sup(I) - (|I| + \inf(J_{j-1})) \\
&= \sup(I) - \sup(I) + \inf(I) - \inf(J_{j-1}) \\
&= \inf(I) - \inf(J_{j-1}) \\
&\geqslant \inf(I) - \sup(J_{j-1}) \\
&= \inf(I - J_{j-1}).
\end{aligned}$$

However, considering 4.3 with j and $j-1$, we have that:

$$\forall v \in J_j,\ (\theta, 1) \models \Phi_1 \tilde{U}_{I-v} \Phi_2 \ \text{ and } \ \forall v \in J_{j-1}, (\theta, 1) \models \Phi_1 \tilde{U}_{I-v} \Phi_2$$

It means that

1. $\forall v \in J_j, \forall k \geqslant 0$ such that $\tau_k \in I - v$:
   either $(\theta, k) \models \Phi_2$, or $\exists 0 \leqslant k' \leqslant k$ such that $(\theta, k') \models \Phi_1$

2. $\forall v \in J_{j-1}, \forall k \geqslant 0$ such that $\tau_k \in I - v$:
   either $(\theta, k) \models \Phi_2$, or $\exists 0 \leqslant k' \leqslant k$ such that $(\theta, k') \models \Phi_1$.

As $(I - J_j) \cap (I - J_{j-1}) \neq \emptyset$,

$$\forall v \in [\sup(J_{j-1}), \inf(J_j)],\ (I - v) \subseteq (I - J_j) \cup (I - J_{j-1}),$$

and so, $\forall v \in [\sup(J_{j-1}), \inf(J_j)]$ and $\forall k \geqslant 0$ such that $\tau_k \in I - v$:

$$\text{either } (\theta, k) \models \Phi_2, \text{ or } \exists 0 \leqslant k' \leqslant k \text{ such that } (\theta, k') \models \Phi_1,$$

which means:

$$\forall v \in [\sup(J_{j-1}), \inf(J_j)],\ (\theta, 1) \models \Phi_1 \tilde{U}_{I-v} \Phi_2.$$

It is in contradiction with 4.4.

We now know that $\forall 2 \leqslant j \leqslant n$, $\inf(J_j) - \inf(J_{j-1}) \geqslant |I|$. So,

$$\inf(J_n) - \inf(J_1) \geqslant (n-1).|I|.$$

As $n > \lceil \frac{\sup(I)}{|I|} \rceil + 1$, we have that

$$\inf(J_n) - \inf(J_1) \ > \ (\lceil \frac{\sup(I)}{|I|} \rceil + 1 - 1)\ .\ |I| \ \geqslant \ \sup(I).$$

It means that $\inf(J_n) - \inf(J_1) > \sup(I)$, and hence $\inf(J_n) > \sup(I)$. It is impossible because the intervals J_j can never entirely exceed $\sup(I)$, because in this case the arc $(\ell, \sigma, x \notin I \wedge x > sup(I))$ can (and must) be taken from (ℓ, J_j) (in way to reach a *minimal* model of the transition function).

It remains to define $M(\Phi)$, using $M(\varphi_{U,I})$ and $M(\varphi_{\tilde{U},I})$. Here is an intuition of how we can do this. By definition of the transitions starting from the initial location $\Phi_{init,\Phi}$, at most one clock copy will be associated with this location (because the initial state is $\{(\ell_0, [0,0])\}$) and it will have no clock copy

associated with this location anymore as soon as clock copies are sent towards other locations. Moreover, all other locations of $\mathcal{A}_\Phi$ are locations associated with subformulas of Φ of type $\Phi_1 U_{I_i} \Phi_2$ or $\Phi_1 \tilde{U}_I \Phi_2$. Let us consider a location of type $\Phi_1 U_{I_i} \Phi_2$, a same reasoning hold for a location of type $\Phi_1 \tilde{U}_I \Phi_2$. We know such a location contains at most $M(\varphi_{U,I})$ clock copies all along π'. Remark that the transition starting from the location of a formula $\Phi_1 U_I \Phi_2$ is $(x.\delta(\Phi_2, \sigma) \wedge x \in I) \vee (x.\delta(\Phi_1, \sigma) \wedge \Phi_1 U_I \Phi_2 \wedge x \leqslant \sup(I))$: it means that $\delta(\Phi_1, \sigma)$ is taken a lot of times while $\delta(\Phi_2, \sigma)$ is only taken once. It is why we must distinguish, in the definition of $M(\Phi)$, the maximal number of clock copies present in configurations reached by π': (1) to verify a subformula φ of Φ that receives a lot of clock copies (denoted $M^1(\varphi)$) (2) to verify a subformula φ of Φ that receives at most one clock copy (denoted $M^\infty(\varphi)$) (3) to verify $\varphi = \Phi$, with the complete automaton $\mathcal{A}_\Phi$ (simply $M(\varphi)$).
Definition 4.10 formally defines $M(\Phi)$. It is not difficult to be convinced that a proof by induction on the structure of Φ enables to show that each configuration of $\mathcal{A}_\Phi$ reached by π' contains at most $M(\Phi)$ clock copies. The only interesting cases of the induction consider subformula of Φ of type $\Phi_1 U_I \Phi_2$ and $\Phi_1 \tilde{U}_I \Phi_2$. We here only develop the case of a subformula φ of Φ of type $\Phi_1 U_I \Phi_2$, the case of a subformula $\Phi_1 \tilde{U}_I \Phi_2$ is similar.

<u>Case $\varphi = \Phi_1 U_I \Phi_2$:</u>

- if $\varphi = \Phi_1 U_I \Phi_2$ can only be reached once, the transition $x.\delta(\Phi_1 U_I \Phi_2, \sigma) = (x.\delta(\Phi_2, \sigma) \wedge x.(x \in I)) \vee (x.\delta(\Phi_1, \sigma) \wedge x.\Phi_1 U_J \Phi_2 \wedge x \leqslant \sup(I))$ must be taken, what creates a loop on $\Phi_1 U_I \Phi_2$. One clock copy is so needed in the location of φ. Moreover, it means that transition $x.\delta(\Phi_1, \sigma)$ can be taken an unbounded number of times, until transition $x.\delta(\Phi_2, \sigma)$ is taken (this will happen a unique time, because it breaks the loop). These transitions correspond to transitions taken (in the same times) when formula Φ_1 is verified an unbounded number of times and to the transition taken when formula Φ_2 is only verified once. The maximal number of clock copies used in this case is so the sum of the maximal numbers of clock copies

needed to verify an unbounded number of times Φ_1 and to verify only once Φ_2, increased by 1: that is $M^1(\varphi) = M^\infty(\Phi_1) + M^1(\Phi_2) + 1$ (induction hypothesis).

- if $\Phi \equiv \varphi = \Phi_1 U_I \Phi_2$, the maximal number of clock copies used by $\mathcal{A}_\Phi$ is the maximum between 1 (the clock copy used by $\varphi_{init,\Phi}$) and the sum of the maximal numbers of clock copies needed to verify an unbounded number of times Φ_1 and to verify only once Φ_2, increased by 1 (because transition '$x.\delta(\Phi_1 U_I \Phi_2, \sigma)$' can only be taken once from $\varphi_{init,\Phi}$, as in the previous case). That is $M(\varphi) = M^\infty(\Phi_1) + M^1(\Phi_2) + 1$ (induction hypothesis).

- if $\Phi \equiv \Phi_1 U_I \Phi_2$ can be verified an unbounded number of times, transition $x.\delta(\Phi_1 U_I \Phi_2, \sigma) = (x.\delta(\Phi_2, \sigma) \wedge x.(x \in I)) \vee (x.\delta(\Phi_1, \sigma) \wedge x.\Phi_1 U_I \Phi_2 \wedge x \leqslant \sup(I))$ can also be taken an unbounded number of times. It means that transitions $x.\delta(\Phi_1, \sigma)$ and $x.\delta(\Phi_2, \sigma)$ can be both taken an unbounded number of times. These transitions correspond to transitions taken (in the same times) when formulas Φ_1 and Φ_2 are verified an unbounded number of times. Moreover, a lot of clock copies can be associated with location $\Phi_1 U_I \Phi_2$, due to the transition '$x.\delta(\Phi_1, \sigma) \wedge x.\Phi_1 U_I \Phi_2 \wedge x \leqslant \sup(I)$' of $x.\delta(\Phi_1 U_I \Phi_2, \sigma)$ (which can be taken an unbounded number of times). Nevertheless, we know the number of clock copies simultaneously present in any location $\Phi_1 U_I \Phi_2$ will never exceed $4.\lceil \frac{\inf(I)}{|I|} \rceil + 2$ thanks to the beginning of the proof of this theorem. We can conclude that the maximal number of clock copies used in this case is the sum of the maximal numbers of clock needed to verify an unbounded number of times Φ_1 and Φ_2, increased by $4.\lceil \frac{\inf(I)}{|I|} \rceil + 2$. That is $M^\infty(\varphi) = M^\infty(\Phi_1) + M^\infty(\Phi_2) + 4.\lceil \frac{\inf(I)}{|I|} \rceil + 2$ (induction hypothesis).

□

Definition 4.10. *Let Φ be an MITL formula in negative normal form. We define $M(\Phi)$, thanks to $M^\infty(\Phi)$ and $M^1(\Phi)$ (intuitively, $M^\infty(\Phi)$ is a bound used for subformulas that can be verified an unbounded number of times while $M^1(\Phi)$*

is a bound used for subformulas that must be verified a unique time) defined as follows:

- *if* $\Phi \equiv \top$ *or* $\Phi \equiv \bot$*, then* $M(\Phi) = 1$ *and* $M^{\infty}(\Phi) = M^{1}(\Phi) = 0$.

- *if* $\Phi \equiv \sigma$ *or* $\Phi \equiv \neg\sigma$*, for* $\sigma \in \Sigma$*, then* $M(\Phi) = 1$ *and* $M^{\infty}(\Phi) = M^{1}(\Phi) = 0$.

- *if* $\Phi \equiv \Phi_1 \wedge \Phi_2$*, then* $M(\Phi) = \max\{1, M^{1}(\Phi_1) + M^{1}(\Phi_2)\}$*,* $M^{\infty}(\Phi) = M^{\infty}(\Phi_1) + M^{\infty}(\Phi_2)$ *and* $M^{1}(\Phi) = M^{1}(\Phi_1) + M^{1}(\Phi_2)$.

- *if* $\Phi \equiv \Phi_1 \vee \Phi_2$*, then* $M(\Phi) = \max\{1, M^{1}(\Phi_1), M^{1}(\Phi_2)\}$*,* $M^{\infty}(\Phi) = \max\{M^{\infty}(\Phi_1), M^{\infty}(\Phi_2)\}$ *and* $M^{1}(\Phi) = \max\{M^{1}(\Phi_1), M^{1}(\Phi_2)\}$.

- *if* $\Phi \equiv \Phi_1 U_I \Phi_2$*, then* $M(\Phi) = M^{\infty}(\Phi_1) + M^{1}(\Phi_2) + 1$*,* $M^{\infty}(\Phi) = \left(4 \times \left\lceil \frac{inf(I)}{|I|} \right\rceil + 2\right) + M^{\infty}(\Phi_1) + M^{\infty}(\Phi_2)$ *and* $M^{1}(\Phi) = M^{\infty}(\Phi_1) + M^{1}(\Phi_2) + 1$.

- *if* $\Phi \equiv \Phi_1 \tilde{U}_I \Phi_2$*, then* $M(\Phi) = M^{1}(\Phi_1) + M^{\infty}(\Phi_2) + 1$*,* $M^{\infty}(\Phi) = \left(2 \times \left\lceil \frac{sup(I)}{|I|} \right\rceil + 2\right) + M^{\infty}(\Phi_1) + M^{\infty}(\Phi_2)$ *and* $M^{1}(\Phi) = M^{1}(\Phi_1) + M^{\infty}(\Phi_2) + 1$.

We suppose by convention that $\frac{inf(I)}{|I|} = 0$ *and* $\frac{sup(I)}{|I|} = 0$ *when* $I =]a, +\infty[$ *or* $[a, +\infty[$ *for a certain* $a \in \mathbb{R}$.

Example 4.11. Let us observe again the OCATA $\mathcal{A}$ of Figure 4.4, corresponding

to the MITL formula $\Phi \equiv \Box\left(a \Rightarrow \left(\Diamond_{[0,1]} b \wedge \Diamond_{[0,1]} c\right)\right)$. We have that:

$$\begin{aligned}
M(\Phi) &= M\left(\Box\left(a \Rightarrow \left(\Diamond_{[0,1]} b \wedge \Diamond_{[0,1]} c\right)\right)\right) \\
&= M\left(\bot \tilde{U}\left(\neg a \vee \left(\top U_{[0,1]} b \wedge \top U_{[0,1]} c\right)\right)\right) \\
&= M^1(\bot) + M^\infty\left(\neg a \vee \left(\top U_{[0,1]} b \wedge \top U_{[0,1]} c\right)\right) + 1 \\
&= 0 + \max(M^\infty(\neg a), M^\infty(\top U_{[0,1]} b \wedge \top U_{[0,1]} c)) + 1 \\
&= \max(0, M^\infty(\top U_{[0,1]} b) + M^\infty(\top U_{[0,1]} c)) + 1 \\
&= \left(\left(4 \times \left\lceil \frac{0}{1} \right\rceil + 2\right) + M^\infty(\top) + M^\infty(b)\right) \\
&\quad + \left(\left(4 \times \left\lceil \frac{0}{1} \right\rceil + 2\right) + M^\infty(\top) + M^\infty(c)\right) + 1 \\
&= 2 + 2 + 1 \quad = \quad 5
\end{aligned}$$

As the state space L of $\mathcal{A}$ contains 3 states $f^\star_\Phi = F^{\max(2.|L|,5)} = F^6$. It means that the F^6-semantics of $\mathcal{A}$ accepts exactly $[\![\Phi]\!]$: $L_{F^6}(\mathcal{A}) = L(\mathcal{A}) = [\![\Phi]\!]$.

4.1.2 Towards a timed automaton

In what precedes, we defined a new semantics for OCATA, called the *interval semantics*. In this semantics, a state of an OCATA $\mathcal{A}$ is a pair (ℓ, I), where ℓ is a location of the OCATA and $I \in \mathcal{I}(\mathbb{R}^+)$. Such intervals are formed using *approximation functions*: they give the possibility of merging intervals associated with a same location in the smaller interval containing them. In general, to use an approximation function f only enables to keep an under-approximation of the language of $\mathcal{A}$, i.e. $L_f(\mathcal{A}) \subseteq L(\mathcal{A})$ and $L^\omega_f(\mathcal{A}) \subseteq L^\omega(\mathcal{A})$ (see Proposition 3.25). Nevertheless, Theorem 4.7 proves that, when we observe an MITL formula Φ and the associated OCATA $\mathcal{A}_\Phi$, there is a approximation function $f^\star_\Phi$ such that $L_{f^\star_\Phi}(\mathcal{A}_\Phi) = L(\mathcal{A}_\Phi)$. Furthermore, this approximation function $f^\star_\Phi$ is *bounded*: it means that we only need to use a *bounded* number of clock copies, all along the runs of $\mathcal{A}_\Phi$, in way to verify if a timed word is in $L(\mathcal{A}_\Phi)$ $(= [\![\Phi]\!])$. Thanks to this bound, the number of states present in a configuration of $\mathcal{A}_\Phi$ is bounded, which

enables to construct a *timed automaton* for any MITL formula: this is the object of this subsection.

Let Φ be an MITL formula, and assume $\mathcal{A}_\Phi = (\Sigma, L^\Phi, \ell_0^\Phi, F^\Phi, \delta^\Phi)$ is its associated OCATA (see Definition 2.164). Let us show how to build a TA $\mathcal{B}_\Phi = (\Sigma, L, \lambda_0, X, F, \delta)$ such that $L(\mathcal{B}_\Phi) = L_{f_\Phi^\star}(\mathcal{A}_\Phi)$. Our construction follows the ideas of the subset construction used in the untimed setting, over finite words, to translate an alternating automaton into an automaton (see Definition 2.46). For the sake of simplicity, in the definition of $\mathcal{B}_\Phi$, we consider that all the intervals used by $\mathcal{A}_\Phi$, even the singular ones, are represented by two different clock copies.

As in the classical subset construction, we could think a location of $\mathcal{B}_\Phi$ is simply a subset of locations of $\mathcal{A}_\Phi$: $L = 2^{L^\Phi}$. However, we must now take into account the *clocks* of $\mathcal{B}_\Phi$. As expected, considering Theorem 4.7, $\mathcal{B}_\Phi$ will have $M(\Phi)$ clocks. Each of the $M(\Phi)$ clock copies used by $\mathcal{A}_\Phi$ is present in a particular location of $\mathcal{A}_\Phi$ and this information must also be retained in way to simulate the runs of $\mathcal{A}_\Phi$ in $\mathcal{B}_\Phi$. Hence, a second idea would be to define each location λ of L as a function from $\lambda : L^\Phi \to 2^X$. Nevertheless, in a configuration of $\mathcal{A}_\Phi$, each of the $M(\Phi)$ clock copies needed is either associated to one particular location or not used at all. So, such a location $\lambda \in L$ should be a function satisfying the following property: $\forall x \in X$, either $\forall \ell \in L^\Phi$, $x \notin \lambda(\ell)$, or there exists a *unique* $\ell \in L^\Phi$ such that $x \in \lambda(\ell)$. Finally, to be precise, there are not clock copies but *intervals*, represented by *pairs of clock copies*, associated to locations in configurations of $\mathcal{A}_\Phi$. The good way to define a location λ of L is so as a function $\lambda : L^\Phi \to 2^{(X^2)}$ satisfying the property that each clock is at most present one time among those in $\bigcup_{\ell \in L^\Phi} \lambda(\ell)$.

Hereunder, we formally define the set of functions enabling to represent locations of $\mathcal{B}_\Phi$. Then, we present the definition of the first components of $\mathcal{B}_\Phi$: its transition function, more complex to define, will be detailed right after. In those definitions, we will often use pairs of clocks to represent the beginning and the end of an interval present in a location of $\mathcal{A}_\Phi$. By convention, we will use letter x to represent the first component of those pairs and so the beginnings of intervals,

while we will use letter y for the second component and, so, the ends of intervals.

Definition 4.12. *We define* $\mathsf{loc}(X)$ *to be set of functions* $\lambda : L^\Phi \to 2^{(X^2)}$ *such that, for all* $(x, y) \in \bigcup_{\ell \in L^\Phi} \lambda(\ell)$*, we have that* $x \neq y$ *and, for all* $(x', y') \in (\bigcup_{\ell \in L^\Phi} \lambda(\ell)) \setminus \{(x, y)\}$*, we have that* $x \neq x'$*,* $x \neq y'$*,* $y \neq x'$ *and* $y \neq y'$*.*

Thanks to the bound $M(\Phi)$ of clock copies needed in the OCATA $\mathcal{A}_\Phi$ (given by Theorem 4.7), we will only need a set X of $M(\Phi)$ clocks in $\mathcal{B}_\Phi$. This way, $\mathsf{loc}(X)$ is a finite set of functions, which will play the role of locations of $\mathcal{B}_\Phi$. Without the bound given by Theorem 4.7, $\mathcal{B}_\Phi$ would have had an infinite number of locations and would not have been a timed automaton. Later, we will mitigate the number of useful locations of $\mathcal{B}_\Phi$ among $\mathsf{loc}(X)$, using the maximal number of intervals that might be associated to each location of $\mathcal{A}_\Phi$ (see the proof of Theorem 4.7).

Definition 4.13. *We define* $\mathcal{B}_\Phi = (\Sigma, L, \lambda_0, X, F, \delta)$*, where:*

- X *is a set of clocks such that* $|X| = M(\Phi)$*,*
- $L = \mathsf{loc}(X)$*,*
- $\lambda_0 \in \mathsf{loc}(X)$ *is such that* $\lambda_0(\ell_0^\Phi) = \{(x, y)\}$*, where* x *and* y *are two clocks arbitrarily chosen from* X*, and* $\lambda_0(\ell) = \varnothing$ *for all* $\ell \in L^\Phi \setminus \{\ell_0^\Phi\}$*.*
- F *is the set of all locations* $\lambda \in L$ *such that* $\{\ell \mid \lambda(\ell) \neq \varnothing\} \subseteq F^\Phi$*.*

Intuitively, a configuration (λ, v) of $\mathcal{B}_\Phi$ encodes the configuration C of $\mathcal{A}_\Phi$ such that for all $\ell \in L^\Phi$: $C(\ell) = \{[v(x), v(y)] \mid (x, y) \in \lambda(\ell)\}$, i.e. the intervals associated to location ℓ of C are given by the values (according to v) of pairs of clocks in $\lambda(\ell)$.

Finally, we must define the set of transitions δ to let $\mathcal{B}_\Phi$ simulate the executions of $\mathcal{A}_\Phi$. We recall (see Remark 3.24) that for each location $\ell \in L^\Phi$, for each $\sigma \in \Sigma$, all *arcs* in δ^Φ are either of the form $(\ell, \sigma, \mathit{true})$, or $(\ell, \sigma, \mathit{false})$, or $\big(\ell, \sigma, \ell \wedge x.(\ell_1 \wedge \cdots \wedge \ell_k) \wedge g\big)$ or of the form $\big(\ell, \sigma, x.(\ell_1 \wedge \cdots \wedge \ell_k) \wedge g\big)$, where g is a *guard* on

x. Let $\lambda \in L$ be a location of $\mathcal{B}_\Phi$, $\ell \in L^\Phi$ and $\sigma \in \Sigma$. Let (x, y) be a pair of clocks occurring in $\lambda(\ell)$ and let us associate to this pair an arc a of δ^Φ of the form (ℓ, σ, γ). Then, we associate to a a *guard* $\mathsf{guard}\,(a)$, and two sets $\mathsf{reset}\,(a)$ and $\mathsf{loop}\,(a)$, defined as follows:

- if $\gamma \in \{\mathit{true}, \mathit{false}\}$, then, $\mathsf{guard}\,(a) = \gamma$ and $\mathsf{reset}\,(a) = \mathsf{loop}\,(a) = \varnothing$.
- if γ is of the form $x.(\ell_1 \wedge \cdots \wedge \ell_k) \wedge g$, then $\mathsf{guard}\,(a) = g$, $\mathsf{reset}\,(a) = \{\ell_1, \ldots, \ell_k\}$ and $\mathsf{loop}\,(a) = \varnothing$.
- if γ is of the form $\ell \wedge x.(\ell_1 \wedge \cdots \wedge \ell_k) \wedge g$, then $\mathsf{guard}\,(a) = g$, $\mathsf{reset}\,(a) = \{\ell_1, \ldots, \ell_k\}$ and $\mathsf{loop}\,(a) = \{(x, y)\}$.

Thanks to those definitions, we can now define δ. Let λ be a location of L, and assume the set of trios (ℓ, x, y) such that $(x, y) \in \lambda(\ell)$ (for $\ell \in L^\Phi$) is denoted:

$$\{(\ell_1, x_1, y_1), \ldots, (\ell_k, x_k, y_k)\}.$$

Remark that several ℓ_i, for $1 \leqslant i \leqslant k$ might be the same location $\ell \in L^\Phi$. Then $(\lambda, \sigma, g, r, \lambda') \in \delta$ iff there is a set $A = \{a_1, \ldots, a_k\}$ of arcs such that:

- for all $1 \leqslant i \leqslant k$: a_i is an arc of δ^Φ of the form $(\ell_i, \sigma, \gamma_k)$, associated with (x_i, y_i).
- For each $\ell \in L^\Phi$, we let $\overline{\lambda}_\ell = \{(x'_1, y'_1)(x'_2, y'_2) \cdots (x'_m, y'_m)\}$ be obtained from $\lambda(\ell)$ by deleting all pairs $(x, y) \notin \bigcup_{1 \leqslant i \leqslant k} \mathsf{loop}\,(a_i)$. We furthermore suppose that (x'_1, y'_1) is the last pair of clock that have been associated to location ℓ. Then, for all $\ell \in L^\Phi$:

  $$\lambda'(\ell) \in \big\{\{(x, y)\} \cup \overline{\lambda}_\ell\,,\ \{(x, y'_1)(x'_2, y'_2) \cdots (x'_m, y'_m)\}\big\} \quad \text{if } \ell \in \bigcup_{1 \leqslant i \leqslant k} \mathsf{reset}\,(a_i)$$

  $$\lambda'(\ell) = \overline{\lambda}_\ell \qquad \text{otherwise}$$

  When $\lambda'(\ell) = \{(x, y)\} \cup \overline{\lambda}_\ell$, we let $R_\ell = \{x, y\}$; when $\lambda'(\ell) = \{(x, y'_1)(x'_2, y'_2) \cdots (x'_m, y'_m)\}$, we let $R_\ell = \{x\}$; and we let $R_\ell = \varnothing$ otherwise.

- $g = \bigwedge_{1 \leqslant i \leqslant k} (\mathsf{guard}\,(a_i)\,[x/x_i] \wedge \mathsf{guard}\,(a_i)\,[x/y_i])$.
- $r = \cup_{\ell \in L^\Phi} R_\ell$.

In what precedes, we denoted by $\mathsf{guard}\,(a_i)\,[x/x']$ the guard $\mathsf{guard}\,(a_i)$ in which x is replaced by x'. In the sequel, such a notation will sometimes be used again.

The following theorem shows that, for all MITL formula Φ, $\mathcal{B}_\Phi$ is a timed automaton recognizing $[\![\Phi]\!]$, as we wanted.

Theorem 4.14. *For each MITL formula Φ, $L(\mathcal{B}_\Phi) = [\![\Phi]\!]$.*

Proof. We prove that $\mathcal{B}_\Phi$ recognizes $[\![\Phi]\!]$ by mapping each configuration of $\mathcal{B}_\Phi$ to a configuration of $\mathcal{A}_\Phi$ and conversely and showing that this mapping is consistent with all runs.

First, let (λ, v) be a configuration of $\mathcal{B}_\Phi$, we map it to the following configuration of $\mathcal{A}_\Phi$. We know that $\forall \ell \in L$, $\lambda(\ell)$ is a set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ of pairs of clocks from X. It corresponds to the (unique) configuration of $\mathcal{A}_\Phi$, $C = \bigcup_{\ell \in L} C(\ell)$ where $C(\ell) = \{[v(x), v(y)] \mid (x, y) \in \lambda(\ell)\}$. It is straightforward to see that, if $(\lambda, v) \overset{t}{\rightsquigarrow} (\lambda', v')$ and (λ, v) is mapped to C, there exists C' such that $C \overset{t}{\rightsquigarrow} C'$ and C' is mapped to (λ', v'). Moreover, we claim that, if $(\lambda, v) \xrightarrow{\sigma} (\lambda', v')$ and (λ, v) is mapped to C, there exists C' such that $C \xrightarrow{\sigma}_{f^\star_\Phi} C'$ and C' is mapped to (λ', v'). This holds because, if $(\lambda, v) \xrightarrow{\sigma} (\lambda', v')$, we can use the arc $a \in \delta^\Phi(\ell, \sigma)$ associated with $(x, y) \in \lambda(\ell)$ to find a minimal model of the state $(\ell, [v(x), v(y)])$ of C, this way, we reach a configuration C' of $\mathcal{A}_\Phi$ that is mapped to (λ', v') thanks to the definition of δ: corresponding clocks are reset in the same time ; we verify the same guards on corresponding clocks ; the configuration we can reach in $\mathcal{B}_\Phi$ corresponds, for each location $\ell \in L$ whose smallest associated interval is [0,0], to group or not this interval with the second associated with ℓ, what corresponds to the configurations of the $f^\star_\Phi$-semantics of $\mathcal{A}_\Phi$ we can reach from C.

Second, let C be a configuration of $\mathcal{A}_\Phi$, we map it to the set of all (λ, v)

such that for all $\ell \in L$: $C(\ell) = \{I_1^\ell, I_2^\ell, \ldots, I_n\}$ iff $v(x_1) = \inf(I_1), v(y_1) = \sup(I_1), \ldots, v(x_n) = \inf(I_n), v(y_n) = \sup(I_n)$. Observe that there are indeed several configurations (λ, v) of $\mathcal{B}_\Phi$ that satisfy this definition: they can all be obtained up to clock renaming. To keep a consistence in our runs, we must only choose the corresponding configuration of $\mathcal{B}_\Phi$ such that: once a pair of clocks is associated with an interval I_j of $C(\ell)$, if I_j is still in $C'(\ell)$, the same clocks represent its bounds. In the same way, when an interval I'_j of the form $[0, \sup(I_j)]$ is in $C'(\ell)$, the same clocks represents its bounds. In contrary, when a new interval I_j $(=[0,0])$ is associated with $C(\ell)$, we can arbitrary choose which unused pair of clocks (x_i, y_i) will represent it (such an unused pair of clocks always exists thanks to Theorem 4.7). Thanks to this trick, we can prove properties similar to those of the first step. □

The following theorem gives an upper bound on the number of locations of $\mathcal{B}_\Phi$.

Proposition 4.15. Let Φ be an MITL formula and $\mathcal{I}_\Phi$ be the set of all the intervals that occur in Φ. $\mathcal{B}_\Phi$ has $M(\Phi)$ clocks and $O((|\Phi|)^{(m.|\Phi|)})$ locations, where $m = \max_{I \in \mathcal{I}_\Phi} \left\{2 \times \left\lceil \frac{\inf(I)}{|I|} \right\rceil + 1, \left\lceil \frac{\sup(I)}{|I|} \right\rceil + 1\right\}$.

Proof. By definition of $\mathcal{B}_\Phi$, $|X| = M(\Phi)$. Moreover, a location of this automaton is an association, to each location ℓ of $\mathcal{A}_\Phi$, of a finite set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ of pairs of clocks from X such that each pair is associated with a unique ℓ. In other words, each couple of clocks (x_i, y_i) can be associated with: either one and only one of the $\ell \in L$ or to no $\ell \in L$. We so have $|L| + 1$ possibilities of association of each pair (x_i, y_i) and we have $\frac{M(\Phi)}{2}$ such pairs. So, $\mathcal{B}_\Phi$ has $(|L| + 1)^{\frac{M(\Phi)}{2}}$ locations, i.e.: $O\left((|\Phi|)^{m.|\Phi|}\right) = O\left(2^{m.|\Phi|.log_2(|\Phi|)}\right)$ (because $|L| = O(|\Phi|)$ and $M(\Phi) = O(2.m.|\Phi|)$). □

Example 4.16. Let us consider again the OCATA $\mathcal{A}_\Phi$ of Figure 4.5, for the MITL formula $\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,2]} b)$. We recall that the initial location Φ_{init} has been removed to enhance readability of the example (this does not modify the

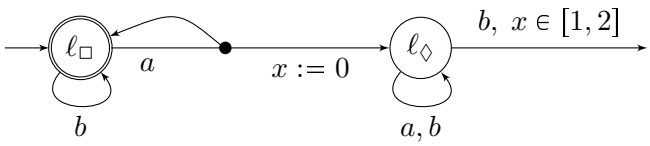


Figure 4.5: OCATA $\mathcal{A}_\Phi$ with $\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,2]} b)$.

accepted language, in the present case). We will show how to construct $\mathcal{B}_\Phi$.
The proof of Theorem 4.7 shows that the number of clock copies given by $M(\Phi)$ is the addition of the number of clock copies needed for each location of $\mathcal{A}_\Phi$ (i.e. subformula of Φ): a location whose outermost operator is U_I will need at most $4.\lceil \frac{\inf(I)}{|I|} \rceil + 2$ clock copies, while a location whose outermost operator is $\tilde{U}_I$ will need at most $2.\lceil \frac{\sup(I)}{|I|} \rceil + 2$ clock copies. For the present formula, location $\ell_\Box$, corresponding to the subformula (in disjunctive normal form) $\bot \tilde{U}_{[0,+\infty[} \left(\neg a \vee \left(\top U_{[1,2]} b\right)\right)$, needs $2.\lceil 0 \rceil + 2 = 2$ clock copies (i.e. 1 interval) and location $\ell_\Diamond$, corresponding to the subformula $\top U_{[1,2]} b$, needs $4.\lceil \frac{1}{1} \rceil + 2 = 6$ clock copies (i.e. 3 intervals). Hence, $\mathcal{B}_\Phi$ has 8 clocks, say $\{x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4\}$. Moreover, we will only consider the locations of the form $\lambda : L^\Phi \to 2^{(X^2)}$ such that: 0 or 1 pair of clocks is associated to $\ell_\Box$, and from 0 to 3 pairs of clocks are associated to $\ell_\Diamond$. The initial location of $\mathcal{B}_\Phi$ is λ_0 such that $\lambda_0(\ell_\Box) = \{(x_1, y_1)\}$ and $\lambda_0(\ell_\Diamond) = \varnothing$. Figure 4.6 represents the automaton $\mathcal{B}_\Phi$. To enhance readability of the example, we do not represent locations of $\mathcal{B}_\Phi$ that could be obtained from a location of the figure by renaming of clocks. For example, location λ_4 such that $\lambda_4(\ell_\Box) = \{(x_1, y_1)\}$ and $\lambda_0(\ell_\Diamond) = \{(x_3, y_3)\}$ is not represented because it is obtained from λ_1 by renaming x_2 in x_3 and y_2 in y_3. When an arc must go from a location present in the figure to a location λ' that is not represented, because it is the renaming of the represented location λ, this arc is drawn as a dotted arrow going to λ. For example, the arc from λ_2 to λ_1 labelled by 'b, $x_2 \in [1,2]$, $y_2 \in [1,2]$' is represented by a dotted line because it should go to λ_4.

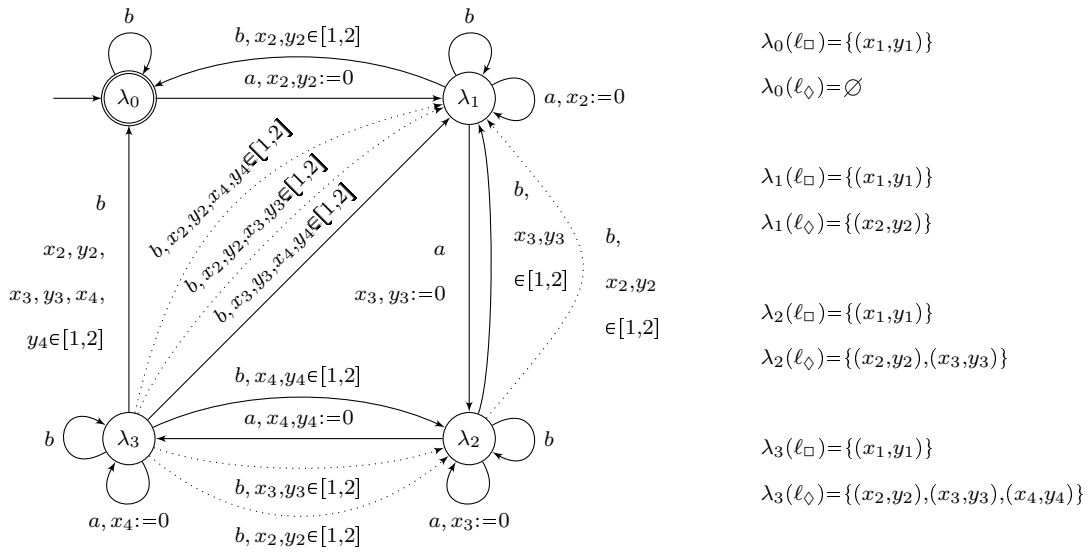


Figure 4.6: The timed automaton $\mathcal{B}_\Phi$.

4.2 MITL model-checking: the techniques

From now on, we fix an MITL formula Φ and assume that the OCATA representing the negation of Φ is $\mathcal{A}_{\neg\Phi} = (\Sigma, L, \ell_0, F, \delta)$. We also fix a TA $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$, and we consider the MITL model-checking problem (see Definition 2.122) and the MITL satisfiability problem (see Definition 2.121) over finite words. The construction of the TA $\mathcal{B}_{\neg\Phi}$ from Φ of the previous section allows to solve those problems using classical algorithms [4] (see subsection 2.4.2). Unfortunately, building $\mathcal{B}_{\neg\Phi}$ can be prohibitive in practice: Proposition 4.15 shows the size of $\mathcal{B}_{\neg\Phi}$ is exponential in the size of $\neg\Phi$ (i.e. in the size of Φ). To mitigate this difficulty, we present an efficient *on the fly* algorithm to perform MITL model-checking, which has as input the TA $\mathcal{B}$ and the OCATA $\mathcal{A}_{\neg\Phi}$ (whose size is linear in the size of Φ).

Our approach follows the steps of [51]. In this paper, Ouaknine and Worrell elaborates techniques to solve the MTL model-checking problem over finite words

(with the pointwise semantics). They construct a timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi}$ that executes $\mathcal{B}$ and $\mathcal{A}_{\neg\Phi}$ in parallel. The aim is then to verify if $L(\mathcal{B}) \subseteq [\![\Phi]\!]$ verifying if $\mathcal{S}_{\mathcal{B},\neg\Phi}$ has an accepting run. However, facing $\mathcal{S}_{\mathcal{B},\neg\Phi}$, they are confronted to two type of infinity. First, this timed transition system is infinitely branching because of the timed successors. Secondly, the number of configurations of $\mathcal{A}_{\neg\Phi}$ is infinite because of the unbounded number of clock copies that can run in parallel in its executions: this induces the infinite depth of $\mathcal{S}_{\mathcal{B},\neg\Phi}$. In their paper, they present appropriate techniques to cope with these two kind of infinity. On the one hand, they use a region abstraction to remove the 'infinitely branching' aspect of $\mathcal{S}_{\mathcal{B},\neg\Phi}$. They proceed representing symbolically each region by a unique word. On the second hand, they have recourse to well quasi order techniques in way we only have to explore a finite depth of $\mathcal{S}_{\mathcal{B},\neg\Phi}$.
We here present our adaptation of their techniques to perform MITL model-checking. We first construct a timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi}$ that executes $\mathcal{B}$ and $\mathcal{A}_{\neg\Phi}$ in parallel: the difference with the work of Ouaknine and Worrell is that we must cope with the interval semantics of $\mathcal{A}_{\neg\Phi}$. Then, the aim will be to verify that $\mathcal{S}_{\mathcal{B},\neg\Phi}$ has no accepting run. We can symmetrically solve the *MITL satisfiability problem* by looking for an accepting run in the timed transition system $\mathcal{S}_{\mathcal{A}_{\neg\Phi},f^{\star}_{\neg\Phi}}$ (see Definition 3.16). Since the techniques are similar for model-checking and satisfiability (see Section 2.4.2), we will only detail the former in this section. While our timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is still infinitely branching because of the timed successors, we can consider it has a finite depth (stopping each of its branches when arriving on a configuration already seen on the same branch). Indeed, thanks to the bound on the number of clock copies needed in $\mathcal{A}_{\neg\Phi}$ (see Theorem 4.7), there exists only finitely many different configurations of $\mathcal{A}_{\neg\Phi}$. To remove the 'infinitely branching' aspect of $\mathcal{S}_{\mathcal{B},\neg\Phi}$, we use a region abstraction in the same spirit as that used in [51]. Our definition of region, as well as the symbolic representation of each region by a unique word, is adapted from that of [51] in way to cope with the intervals.

We start by formally defining $\mathcal{S}_{\mathcal{B},\neg\Phi}$, representing the parallel execution of $\mathcal{B}$ and $\mathcal{A}_{\neg\Phi}$.

Definition 4.17. *Let Φ be an MITL formula and $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$ be a timed automaton. We define the timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi} = (\Sigma, S, s_0, \rightsquigarrow, \rightarrow, \alpha)$ where:*

- *S is the set of pairs of the form (B, C), for configurations $B = (b, v)$ of $\mathcal{B}$ and C of $\mathcal{A}_{\neg\Phi}$,*
- *$s_0 = ((b_0, v_0), \{(\ell_0, [0, 0])\})$, where v_0 is the valuation such that $v_0(x) = 0$, $\forall x \in X$,*
- *α contains all the elements (B, C) of S such that B is accepting for $\mathcal{B}$ and C is accepting for $\mathcal{A}_{\neg\Phi}$,*
- *the transition relation $\rightsquigarrow$ takes care of the elapsing of time: $\forall t \in \mathbb{R}^+$, $(B, C) \overset{t}{\rightsquigarrow} (B', C')$ iff $(B', C') = (B + t, C + t)$ and $\rightsquigarrow \; = \bigcup_{t \in \mathbb{R}^+} \overset{t}{\rightsquigarrow}$,*
- *the transition relation $\rightarrow$ takes care of discrete transitions between locations: $(B, C) \overset{\sigma}{\rightarrow} (B', C')$ iff $B \overset{\sigma}{\rightarrow} B'$ in $\mathcal{B}$ and $C \overset{\sigma}{\rightarrow}_{f^\star_{\neg\Phi}} C'$ in $\mathcal{A}_{\neg\Phi}$. We have $\rightarrow \; = \bigcup_{\sigma \in \Sigma} \overset{\sigma}{\rightarrow}$.*

Example 4.18. Let us consider the TA $\mathcal{B}$ of Figure 4.8 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 4.7, for the MITL formula $\neg\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,3]} b)$. Figure 4.9 gives a part of the timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi}$. The elements of α are represented with a double border. We are not able to represent all the transitions $\overset{t}{\rightsquigarrow}$ for $t \in \mathbb{R}$, so that we represent most of them by a unique arc indexed by a part of $\mathbb{R}$. Nevertheless, one must keep in mind that $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is in fact infinitely branching. The transitions leading to the states underneath the figure come from the fact that, in $\mathcal{A}_{\neg\Phi}$, the image of configuration $C = \{(\ell_\Box, 1), (\ell_\Diamond, 0), (\ell_\Diamond, 1)\}$ by $f^\star_{\neg\Phi}$ is $\{\{(\ell_\Box, 1), (\ell_\Diamond, 0), (\ell_\Diamond, 1)\}, \{(\ell_\Box, 1), (\ell_\Diamond, [0, 1])\}\}$.

The following proposition holds by construction of $\mathcal{S}_{\mathcal{B},\neg\Phi}$:

Proposition 4.19. For every MITL formula Φ, the associated OCATA $\mathcal{A}_{\neg\Phi}$ and function $f^\star_{\neg\Phi}$, and for every timed automaton $\mathcal{B}$: $L(\mathcal{S}_{\mathcal{B},\neg\Phi}) = L_{f^\star_{\neg\Phi}}(\mathcal{A}_{\neg\Phi}) \cap L(\mathcal{B})$.

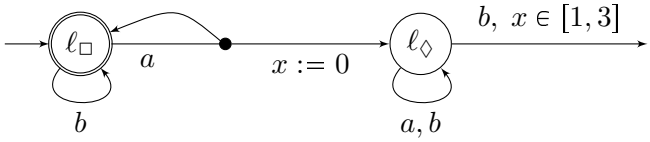


Figure 4.7: OCATA $\mathcal{A}_{\neg\Phi}$ with $\neg\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,3]} b)$.

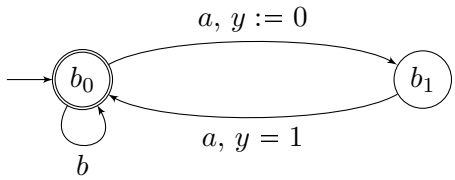


Figure 4.8: A timed automaton $\mathcal{B}$.

From $\mathcal{S}_{\mathcal{B},\neg\Phi}$, we define a region abstraction coping with the intervals. In the sequel, we note $\mathrm{c}_{\max}$ the maximal constant of automata $\mathcal{B}$ and $\mathcal{A}_{\neg\Phi}$.

Definition 4.20. *We define the equivalence relation $\sim$ on $\mathbb{R}^+$ by:*

$$u \sim v \quad \textit{iff} \quad \begin{array}{l} \textit{either } u > c_{\max} \textit{ and } v > c_{\max}, \\ \textit{or } u \leqslant c_{\max} \textit{ and } v \leqslant c_{\max},\ \lceil u \rceil = \lceil v \rceil \textit{ and } \lfloor u \rfloor = \lfloor v \rfloor. \end{array}$$

The set of classes of $\sim$, called regions, is $REG = \{\{0\},]0,1[, \{1\},]1,2[, \ldots,]c_{\max}-1, c_{\max}[, \{c_{\max}\},]c_{\max}, +\infty[\}$[2]. We will note $Reg(v)$ the class of $v \in \mathbb{R}^+$.

In the sequel, $\forall v \in \mathbb{R}^+$, we note $frac(v)$ the fractional part of v. Moreover, for technical reasons, we suppose that $\forall v > c_{\max}, frac(v) = 0$: intuitively, we are not interested in the fractional part values beyond $c_{\max}$.

We can now define an equivalence relation $\equiv$ on S. In its formal definition hereunder, the states of the two configurations of $\mathcal{A}_{\neg\Phi}$ are indexed on a same set K in aim to make 'correspond' the states of same index. For the two states (B, C) and (B', C') of S, we only want that $(B, C) \equiv (B', C')$ if all the clock

[2]These are classical regions used for a timed automaton with one clock.

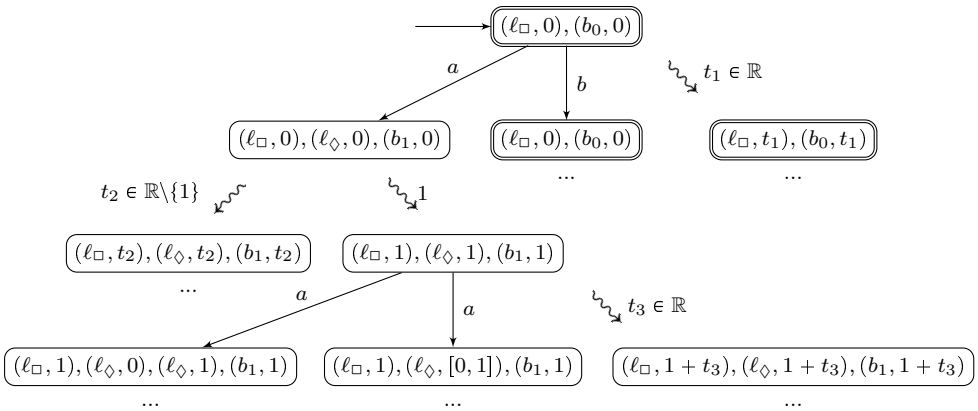


Figure 4.9: Representation of a part of $\mathcal{S}_{\mathcal{B},\neg\Phi}$.

values present in (B, C) (values of infima and suprema of intervals, values of clocks of $\mathcal{B}$) present the same fractional part order as the clock values present in (B', C'), according to indices. The length of the following definition is due to this condition which is divided in 6 items (conditions 3. to 8.) comparing: (i) values of clocks of $\mathcal{B}$, (ii) values of infima (of intervals of $\mathcal{A}_{\neg\Phi}$), (iii) values of suprema, (iv) a value of an infimum with a value of a supremum, (v) a value of an infimum with a value of a clock of $\mathcal{B}$, (vi) a value of a supremum with a value of a clock of $\mathcal{B}$.

Definition 4.21. *Let Φ be an MITL formula, $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$ be a timed automaton and $\mathcal{S}_{\mathcal{B},\neg\Phi}$, of state space S, be the transition system given by Definition 4.17. Let K be a set of indices and $(B, C), (B', C') \in S$, with $B = (b, v)$, $B' = (b', v')$, $C = \{(\ell_k, I_k)_{k\in K}\}$ and $C' = \{(\ell'_k, I'_k)_{k\in K}\}$.*
Then, we define $(B, C) \equiv (B', C')$ iff:

1. *$b = b'$ and $\forall k \in K$, $\ell_k = \ell'_k$,*
2. *$\forall 1 \leqslant p \leqslant n : v(x_p) \sim v'(x_p)$ and $\forall k \in K : (\inf(I_k) \sim \inf(I'_k) \wedge \sup(I_k) \sim \sup(I'_k))$,*
3. *$\forall 1 \leqslant p, q \leqslant n : frac(v(x_p)) \bowtie frac(v(x_q))$ iff $frac(v'(x_p)) \bowtie frac(v'(x_q))$,*

4. $\forall k, k' \in K : frac(\inf(I_k)) \bowtie frac(\inf(I_{k'}))$ *iff* $frac(\inf(I'_{k'})) \bowtie frac(\inf(I'_{k'}))$,

5. $\forall k, k' \in K : frac(\sup(I_k)) \bowtie frac(\sup(I_{k'}))$ *iff* $frac(\sup(I'_k)) \bowtie frac(\sup(I'_{k'}))$,

6. $\forall k, k' \in K : frac(\inf(I_k)) \bowtie frac(\sup(I_{k'}))$ *iff* $frac(\inf(I'_k)) \bowtie frac(\sup(I'_{k'}))$,

7. $\forall k \in K, \forall 1 \leqslant p \leqslant n : frac(\inf(I_k)) \bowtie frac(v(x_p))$ *iff* $frac(\inf(I'_k)) \bowtie frac(v'(x_p))$,

8. $\forall k \in K, \forall 1 \leqslant p \leqslant n : frac(\sup(I_k)) \bowtie frac(v(x_p))$ *iff* $frac(\sup(I'_k)) \bowtie frac(v'(x_p))$,

where $\bowtie \in \{<, =, >\}$.

Condition 1. forces corresponding (ℓ_k, I_k) and (ℓ'_k, I'_k), as well as the two configurations of $\mathcal{B}$, to have (respectively) the same locations. Condition 2. forces intervals of corresponding (ℓ_k, I_k) and (ℓ'_k, I'_k) to have their infima and suprema in the same regions (i.e. classes of $\sim$), and values of the same clocks of the two configurations of $\mathcal{B}$ to be in the same region. The other conditions forces all the clock values present in (B, C) (values of infimum and supremum of intervals, values of clocks of $\mathcal{B}$) to present the same fractional part order as the corresponding clock values present in (B', C').

Example 4.22. We consider again the TA $\mathcal{B}$ of Figure 4.8 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 4.7, for the MITL formula $\neg\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,3]} b)$. We consider the following states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$:

$$\begin{aligned} s_1 &:= \{(\ell_\Box, 1.4), (\ell_\Diamond, [0, 0.2]), (b_1, 0.7)\} \\ s_2 &:= \{(\ell_\Box, 1.6), (\ell_\Diamond, [0, 0.5]), (b_1, 0.65)\} \\ s_3 &:= \{(\ell_\Box, 1.8), (\ell_\Diamond, [0, 0.5]), (b_1, 0.7)\}. \end{aligned}$$

On the one hand, we have $s_1 \equiv s_2$. Indeed, we make correspond $(\ell_\Box, 1.4)$ with $(\ell_\Box, 1.6)$, $(\ell_\Diamond, [0, 0.2])$ with $(\ell_\Diamond, [0, 0.5])$ and $(b_1, 0.7)$ with $(b_1, 0.65)$ in way condition 1. is satisfied. Condition 2. is satisfied because $1.4 \sim 1.6$ (class $]1, 2[$), $0 \sim 0$

(class $\{0\}$), $0.2 \sim 0.5$ (class $]0,1[$) and $0.7 \sim 0.65$ (class $]0,1[$). Sorting the clock values of s_1 following the increasing order of their fractional parts, we obtain:

$$frac(0) < frac(0.2) < frac(1.4) < frac(0.7).$$

In s_2, we obtain:

$$frac(0) < frac(0.5) < frac(1.6) < frac(0.65).$$

As corresponding clocks values are in the same place in these orderings, one can verify that conditions 3. to 8. are indeed satisfied. For instance, condition 7. holds because, for each $\bowtie \in \{<, =, >\}$:

$$\begin{array}{rcl} frac(1.4) = 0.4 \bowtie frac(0.7) = 0.7 & \text{iff} & frac(1.6) = 0.6 \bowtie frac(0.65) = 0.65, \text{ and} \\ frac(0) = 0 \bowtie frac(0.7) = 0.7 & \text{iff} & frac(0) = 0 \bowtie frac(0.65) = 0.65. \end{array}$$

On the second hand, $s_1 \not\equiv s_3$. Indeed, we must make correspond $(\ell_\Box, 1.4)$ with $(\ell_\Box, 1.8)$, $(\ell_\Diamond, [0, 0.2])$ with $(\ell_\Diamond, [0, 0.5])$ and $(b_1, 0.7)$ with $(b_1, 0.7)$ to satisfy condition 1.. This way, condition 2. is also satisfied. Nevertheless, sorting the clock values of s_1 following the increasing order of their fractional parts, we obtain

$$frac(0) < frac(0.2) < frac(1.4) < frac(0.7);$$

while, in s_3, we have

$$frac(0) < frac(0.5) < frac(0.7) < frac(1.8).$$

Corresponding clocks values are *not* in the same place in these orderings, so that conditions among 3. to 8. will not be satisfied. For instance, condition 7. is not satisfied because $frac(1.4) = 0.4 < frac(0.7) = 0.7$ while $frac(1.8) = 0.8 > frac(0.7) = 0.7$.

We now present a proposition stating that the equivalence relation $\equiv$ induces a time-abstract bisimulation on the states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$. This bisimulation will enable us to elaborate an MITL model-checking algorithm only observing the classes of $\equiv$ (we will see they are in finite number) instead of exploring all the states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ (in infinite number). The proof of this proposition can be found in the appendix.

Proposition 4.23 (Time-abstract bisimulation)**.** Let Φ be an MITL formula, $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$ be a timed automaton and $\mathcal{S}_{\mathcal{B},\neg\Phi}$, of state space S, be the transition system given by Definition 4.17. Let $(B_1, C_1), (B_2, C_2) \in S$ such that $(B_1, C_1) \equiv (B_2, C_2)$. Then:

1. for each transition
   $(B_1, C_1) \overset{t}{\rightsquigarrow} (A_1, D_1)$ with $t \in \mathbb{R}^+$ and $(A_1, D_1) \in S$,
   there exists $t' \in \mathbb{R}^+$ and $(A_2, D_2) \in S$ such that:
   $(B_2, C_2) \overset{t'}{\rightsquigarrow} (A_2, D_2)$ and $(A_1, D_1) \equiv (A_2, D_2)$;

2. for each transition
   $(B_1, C_1) \xrightarrow{\sigma} (A_1, D_1)$, with $\sigma \in \Sigma$ and $(A_1, D_1) \in S$,
   there exists $(A_2, D_2) \in S$ such that:
   $(B_2, C_2) \xrightarrow{\sigma} (A_2, D_2)$ and $(A_1, D_1) \equiv (A_2, D_2)$.

Remark that the size of the configurations of $\mathcal{A}_{\neg\Phi}$ we can encounter in $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is bounded by $M(\neg\Phi)$, thanks to the use of $f^{\star}_{\neg\Phi}$. So, there is only a finite number of such configurations. As a consequence, the number of configurations of $\mathcal{A}_{\neg\Phi}$ we can encounter in $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is finite. As the number of regions is also finite, the quotient of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ by $\equiv$ is finite and we can elaborate a model-checking algorithm using it. In the sequel, we will define a symbolic representation of each of these regions by a unique word. Before giving the formal definition, let us explain how we proceed on an example.

Example 4.24. Let us consider the TA $\mathcal{B}$ of Figure 4.8 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 4.7. To make this example interesting, we consider the following unreachable state of $\mathcal{S}_{\mathcal{B},\neg\Phi}$:

$$s_1 \ := \ \{(b_1, 1.5), (\ell_\Diamond, [0, 0.5]), (\ell_\Diamond, [1.7, 3.3])\}.$$

To represent the region of s_1 according to $\equiv$, we are interested in each of the values of clocks and clock copies 1.5, 0, 0.5, 1.7 and 3.3, as well as in the order of their fractional parts. Furthermore, it is important to remember that values 0

and 0.5 form an interval, as well as values 1.7 and 3.3.
We first construct a 3-tuple to represent each value of the clock copies of $\mathcal{A}_{\neg\Phi}$. A 3-tuple contains the location to which the clock copy is associated, the value of the clock copy, and a number to remember which couples of clock copies form intervals. For s_1, we have:

$$\begin{array}{llll} \text{value} & 0 & \text{is represented by} & (\ell_\Diamond, 0, 1), \\ \text{value} & 0.5 & \text{is represented by} & (\ell_\Diamond, 0.5, 1), \\ \text{value} & 1.7 & \text{is represented by} & (\ell_\Diamond, 1.7, 2), \\ \text{value} & 3.3 & \text{is represented by} & (\ell_\Diamond, 3.3, 2). \end{array}$$

The value of clocks of $\mathcal{B}$ are also represented by a 3-tuple containing the location in which $\mathcal{B}$ is, the value of the nth clock of $\mathcal{B}$ and a marker n to remember to which clock of $\mathcal{B}$ the value corresponds. For s_1, we have:

$$\text{value} \quad 1.5 \quad \text{is represented by} \quad (b_1, 1.5, 1).$$

Secondly, we sort all the obtained 3-tuples in different sets: we create a set for each value of the fractional parts of the clock values. We recall that the fractional part of clock (copies) values beyond c_{max} (= 3, here) is considered to be 0. For our example, we have:

$$\begin{array}{l} \{(b_1, 1.5, 1), (\ell_\Diamond, 0.5, 1)\} \\ \{(\ell_\Diamond, 0, 1), (\ell_\Diamond, 3.3, 2)\} \\ \{(\ell_\Diamond, 1.7, 2)\} \end{array}$$

When the values of clocks and clock copies will be replaced by the region of REG they are in, each of these sets will be a *letter* of the word symbolically representing the region of s_1 according to $\equiv$. Nevertheless, before replacing the values by the good regions of REG, we must sort the letters of this word. In fact, we sort these letters according to the order of the fractional part of clock values they represent. We here obtain:

$$\{(\ell_\Diamond, 0, 1), (\ell_\Diamond, 3.3, 2)\}\ \{(b_1, 1.5, 1), (\ell_\Diamond, 0.5, 1)\}\ \{(\ell_\Diamond, 1.7, 2)\}$$

The region of s_1 is so symbolically represented by:

$$\{(\ell_\Diamond, \{0\}, 1), (\ell_\Diamond,]3, +\infty[, 2)\}\ \{(b_1,]1, 2[, 1), (\ell_\Diamond,]0, 1[, 1)\}\ \{(\ell_\Diamond,]1, 2[, 2)\}$$

Let us note $\max_\ell$ the maximal number of intervals that can be present in location $\ell \in L$ of $\mathcal{A}_{\neg\Phi}$ (given by the proof of Theorem 4.7), and $n := |X|$ the number of clocks of $\mathcal{B}$. Then, the alphabet on which are written the words symbolically representing the region of each $s \in S$ is denoted Λ and consists in the subsets of:

$$(B \cup L) \times REG \times \{1, 2, \ldots, \max(\max_{\ell \in L}(\max_\ell), n)\}.$$

We now formally define the function $H : S \to \Lambda^\star$ that associates with each $s \in S$ the region it is in.

Definition 4.25. *For $s = \{(\ell_k, I_k)_{k \in K}\} \cup \{(b, v)\}$, $H(s) = H_1 H_2 \cdots H_p$ is defined as follows:*

1. *For each location ℓ, let $C(\ell) = \{(\ell', I) \in C \mid \ell' = \ell\}$. Assume $C(\ell) = \{(\ell_1, I_1), \ldots, (\ell_k, I_k)\}$, with $I_1 \leqslant \cdots \leqslant I_k$. Then, we first build $E_\ell = \{(\ell_i, \inf(I_i), i), (\ell_i, \sup(I_i), i) \mid 1 \leqslant i \leqslant k\}$.*

2. *We treat $(\ell^{\mathcal{B}}, v)$ symmetrically, and let $E^{\mathcal{B}} = \{(\ell^{\mathcal{B}}, v(x_1), 1), \ldots, (\ell^{\mathcal{B}}, v(x_n), m, n)\}$. We let $\mathcal{E} = E^{\mathcal{B}} \cup_{\ell \in L} E_\ell$. That is, all elements in $\mathcal{E}$ are tuples (ℓ, v, i), where ℓ is a location (of $\mathcal{A}_\Phi$ or $\mathcal{B}$), v is a real value (interval endpoint or clock value) and i is bookkeeping information that links v to an interval (if ℓ is a location of $\mathcal{A}_\Phi$), or to a clock (if ℓ is a location of $\mathcal{B}$).*

3. *We partition $\mathcal{E}$ into $\mathcal{E}_1, \ldots, \mathcal{E}_p$ such that each $\mathcal{E}_i$ contains all elements from $\mathcal{E}$ with the same fractional part of their second component (recall that we assume $frac(u) = 0$ for all $u > c_{max}$). We assume the ordering $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_p$ reflects the increasing ordering of the fractional parts.*

4. *For all $1 \leqslant i \leqslant p$, we obtain H_i from $\mathcal{E}_i$ by replacing the second component of all elements in $\mathcal{E}_i$ by the region from REG they belong to.*

The following proposition shows that the words obtained from states of S using function H give a correct representation of classes of $\equiv$.

Proposition 4.26. Let $s, s' \in S$. We have: $s \equiv s'$ iff $H(s) = H(s')$.

Proof. ($\Rightarrow$) Suppose that $s \equiv s'$, then the order of the fractional parts of all the clock values that s contains is the same than those of all the corresponding clock values that s' contains (see $\equiv$ conditions 3. to 8.). Moreover, their corresponding states have their infima (respectively their suprema, respectively clocks values) in the same region (see $\equiv$ condition 2.) and their locations are the same. The way $H(s)$ and $H(s')$ are constructed, their will be no difference between these two words.
($\Leftarrow$) Suppose $H(s) = H(s')$, then we associate each interval of the configuration s of $\mathcal{A}_{\neg\Phi}$, clearly defined by two 3-tuples in $H(s)$, to the interval of s' that is represented by the two corresponding 3-tuples of $H(s')$. Moreover, for $1 \leqslant i \leqslant n$, we associate each value v_i of the clock x_i of $\mathcal{B}$, clearly defined by a certain 3-tuple in $H(s)$, with the value of v'_i of the clock x_i of $\mathcal{B}$ represented in the corresponding 3-tuple of $H(s')$. As $H(s) = H(s')$, conditions 1. and 2. of $\equiv$ are of course verified. The other conditions are also respected thanks to the groupings executed on the elements of $H(s)$ and $H(s')$ to reflect the increasing order of the fractional parts of the second components (i.e. clock) values. $\square$

Thanks to the bisimulation lemma, instead of considering all the states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$, in the following, we will be only interested in the classes of $\equiv$ in the states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ (which are the words given by function H). We here define the set of classes of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ we are interested in and the (timed and discrete) transitions between those classes.

Definition 4.27. *We define:*

$$\mathcal{H} \;=\; \mathcal{S}_{\mathcal{B},\neg\Phi}/\equiv \;=\; \{H(s) \mid s \in S\}.$$

For all $W^1, W^2 \in \mathcal{H}$ *and* $\sigma \in \Sigma$ *we define* $W^1 \xrightarrow{\sigma} W^2$ *iff* $\forall s^1 \in H^{-1}(W^1)$, $\exists s^2 \in H^{-1}(W^2)$ *such that* $s^1 \xrightarrow{\sigma} s^2$.
For all $W^1, W^2 \in \mathcal{H}$, *we define* $W^1 \longrightarrow_T W^2$ *iff* $\forall s^1 \in H^{-1}(W^1)$, $\exists t \in \mathbb{R}$ *and* $\exists s^2 \in H^{-1}(W^2)$ *such that* $s^1 \overset{t}{\rightsquigarrow} s^2$.

Example 4.28. Once again, we consider again the TA $\mathcal{B}$ of Figure 4.8 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 4.7. Let us consider the state $\{(\ell_\Box, 2.2), (\ell_\Diamond, [1,2]), (b_0, 2)\}$ of $\mathcal{S}_{\mathcal{B},\neg\Phi}$. Its class in $\mathcal{H}$ is the word:

$$W^1 \ := \ \{(\ell_\Diamond, \{1\}, 1), (\ell_\Diamond, \{2\}, 1), (b_0, \{2\}, 1)\}\{(\ell_\Box,]2, 3[, 1)\}.$$

Let us note:

$$W^2 \ := \ \{(b_0, \{2\}, 1)\}\{(\ell_\Box,]2, 3[, 1)\}.$$

We have $W^1 \xrightarrow{b} W^2$. Indeed, let $s^1 \in H^{-1}(W^1)$: s^1 will be of the form $\{(\ell_\Box, t), (\ell_\Diamond, [1,2]), (b_0, 2)\}$, for a certain $t \in]2,3[$. Let us consider $s^2 = \{(\ell_\Box, t), (b_0, 2)\}$, we have that $s^2 \in H^{-1}(W^2)$ and $s^1 \xrightarrow{b} s^2$.
Now, let us note:

$$W^3 \ := \ \{(\ell_\Diamond,]1, 2[, 1), (\ell_\Diamond,]2, 3[, 1), (b_0,]2, 3[, 1)\}\{(\ell_\Box,]2, 3[, 1)\}.$$

We have $W^1 \longrightarrow_T W^3$. Indeed, let $s^1 \in H^{-1}(W^1)$: s^1 will be of the form $\{(\ell_\Box, t), (\ell_\Diamond, [1,2]), (b_0, 2)\}$, for a certain $t \in]2,3[$. Let us consider $t' \in]0,1[$ and $s^3 = \{(\ell_\Box, t+t'), (\ell_\Diamond, [1+t', 2+t']), (b_0, 2+t')\}$ such that $t+t' \in]2,3[$ (such a t' always exists as we are using open intervals of $\mathbb{R}$). Hence, we have $s^3 \in H^{-1}(W^3)$ and $s^1 \overset{t'}{\rightsquigarrow} s^3$.

We defined that $W^1 \xrightarrow{\sigma} W^2$ iff **for all** $s^1 \in H^{-1}(W^1)$:

$$\exists s^2 \in H^{-1}(W^2) \text{ such that } s^1 \xrightarrow{\sigma} s^2. \tag{4.5}$$

The following proposition shows that no matter the choice of $s^1 \in H^{-1}(W^1)$: indeed, if property 4.5 is true for *one* such s^1, it will consequently be true for *all* such s^1.

Proposition 4.29. Let $W^1, W^2 \in \mathcal{H}$, $\sigma \in \Sigma$ and $t \in \mathbb{R}^+$.
$W^1 \xrightarrow{\sigma} W^2$ iff $\exists s^1 \in H^{-1}(W^1)$ and $s^2 \in H^{-1}(W^2) : s^1 \xrightarrow{\sigma} s^2$.

Proof. ($\Rightarrow$) Follows directly from Definition 4.27.
($\Leftarrow$) Suppose that $s^1 \in H^{-1}(W^1), s^2 \in H^{-1}(W^2)$, and that $s^1 \xrightarrow{\sigma} s^2$. Let $s^3 \in H^{-1}(W^1)$, we must prove that $\exists s^4 \in H^{-1}(W^2) : s^3 \xrightarrow{\sigma} s^4$. As $s^3 \in H^{-1}(W^1)$ and $s^1 \in H^{-1}(W^1)$, by Proposition 4.26, $s^3 \equiv s^1$. As $s^1 \xrightarrow{\sigma} s^2$, Proposition 4.23 ensures that $\exists s^4 \in H^{-1}(W^2) : s^3 \xrightarrow{\sigma} s^4$. □

As previously explained, our model-checking algorithm will consist in looking for a path to an accepting state of $\mathcal{H}$ (corresponding to a class of accepting states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$). We will conclude that $\mathcal{B} \not\models \Phi$ if and only if such a path exists. We so define $Post(\mathcal{W})$, for $\mathcal{W} \subseteq \mathcal{H}$: this operator will be used in our algorithm to compute the successors of the set of states of $\mathcal{H}$ we have already reached.

Definition 4.30. *Let $\mathcal{W} \subseteq \mathcal{H}$, we define:*

$$Post(\mathcal{W}) := \{W' \in \mathcal{H} \mid \exists \sigma \in \Sigma,\ W \in \mathcal{W} \text{ and } W'' \in \mathcal{H} : W \longrightarrow_T W'' \xrightarrow{\sigma} W'\}.$$

To ensure the termination of our following algorithm, we need $Post(W)$ to be finite and effectively computable, for any word $W \in \mathcal{H}$. This is what claims the following proposition. In the same time, the proof of this proposition gives a procedure to compute $Post(W)$ from W.

Proposition 4.31. For each word $W \in \mathcal{H}$, $Post(W)$ is finite and effectively computable.

Proof. Let $W \in \mathcal{H}$. The set of all W'' such that $W \longrightarrow_T W''$ is a finite set of words with the same number of 3-tuples than W. We form this set accumulating the words computed recursively as follows, using at each time the last W_{next} obtained (and starting from W):

- if the first letter of W_{next} contains 3-tuples whose second component is $\{0\}$ or $\{1\}$ or ...or $\{c_{\max}\}$, the following W_{next} is the word created as follows:

  1. the 3-tuples of this first letter whose second component is $\{c_{\max}\}$ are replaced by the same 3-tuples in which $\{c_{\max}\}$ is replaced by $]c_{\max}, +\infty[$,
  2. the other 3-tuples of this first letter are deleted from it (if it then becomes empty, it is omitted). A new set of 3-tuples is created as a new second letter: it will contain these same 3-tuples in which the second component is replaced by the immediately following region ($]0,1[$ instead of $\{0\}$, $]1,2[$ instead of $\{1\}, \ldots,]c_{\max}-1, c_{\max}[$ instead of $\{c_{\max}-1\}$). The end of the word does not change.

- else, the following W_{next} is the word created as follows. The last letter of W_{next} is deleted. Its 3-tuples are modified to create a new set that will contain these same 3-tuples in which the second component is replaced by the immediately following region ($\{1\}$ instead of $]0,1[$, $\{2\}$ instead of $]1,2[, \ldots, \{c_{\max}\}$ instead of $]c_{\max}-1, c_{\max}[$). This new set is either joined with the first letter of the modified W_{next}, if it contains 3-tuples having $]c_{\max}, +\infty[$ as second components, or added as a new first letter of the modified W_{next} otherwise. The rest of the word does not change.

We stop when we encounter a W_{next} that has a unique letter whose 3-tuples have $]c_{\max}, +\infty[$ as second components.
Then, for each possible W'' such that $W \longrightarrow_T W''$, we easily find a $s \in S$ such that $H(s) = W''$ (note that the choice of s does not matter thanks to Proposition 4.29). For all $\sigma \in \Sigma$, it is easy to compute the set of elements s' of S such that $s \xrightarrow{\sigma} s'$. This set is finite and, from each of its elements s', we can get back $H(s')$.
Once we have examined each letter $\sigma \in \Sigma$, for each possible W'', the (finite !) set of all the $H(s')$ found form $Post(W)$. □

Example 4.32. Once again, we consider again the TA $\mathcal{B}$ of Figure 4.8 and the

OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 4.7. Let us consider the following word of $\mathcal{H} = \mathcal{S}_{\mathcal{B},\neg\Phi}/\equiv$:

$$W := \{(\ell_\Box,]3, +\infty[, 1), (\ell_\Diamond, \{2\}, 1), (\ell_\Diamond, \{3\}, 1), (b_0, \{3\}, 1)\}.$$

We will look for $Post(W)$. The procedure of the proof of Proposition 4.31 recursively gives the following timed successors of W:

$$\begin{aligned}
W &:= \{(\ell_\Box,]3, +\infty[, 1), (\ell_\Diamond, \{2\}, 1), (\ell_\Diamond, \{3\}, 1), (b_0, \{3\}, 1)\},\\
W^1 &:= \{(\ell_\Box,]3, +\infty[, 1), (\ell_\Diamond,]3, +\infty[, 1), (b_0,]3, +\infty[, 1)\}\{(\ell_\Diamond,]2, 3[, 1)\},\\
W^2 &:= \{(\ell_\Box,]3, +\infty[, 1), (\ell_\Diamond,]3, +\infty[, 1), (b_0,]3, +\infty[, 1), (\ell_\Diamond, \{3\}, 1)\},\\
W^3 &:= \{(\ell_\Box,]3, +\infty[, 1), (\ell_\Diamond,]3, +\infty[, 1), (b_0,]3, +\infty[, 1), (\ell_\Diamond,]3, +\infty[, 1)\}.
\end{aligned}$$

In way to compute the discrete successors of W, we might consider $s = \{(\ell_\Box, 3.1), (\ell_\Diamond, [2, 3]), (b_0, 3)\}$, as $H(s) = W$. Let us note:

$$\begin{aligned}
s^1 &:= \{(\ell_\Box, 3.1), (\ell_\Diamond, 0), (\ell_\Diamond, [2, 3]), (b_1, 0)\},\\
s^2 &:= \{(\ell_\Box, 3.1), (\ell_\Diamond, [0, 3]), (b_1, 0)\},\\
s^3 &:= \{(\ell_\Box, 3.1), (\ell_\Diamond, [2, 3]), (b_0, 3)\},\\
s^4 &:= \{(\ell_\Box, 3.1), (b_0, 3)\}.
\end{aligned}$$

We have that $s \xrightarrow{a} s^1$, $s \xrightarrow{a} s^2$, $s \xrightarrow{b} s^3$ and $s \xrightarrow{b} s^4$. This gives the following elements of $Post(W)$:

$$\begin{aligned}
W^1_{post} &:= \{(\ell_\Box,]3, +\infty[, 1), (\ell_\Diamond, \{0\}, 1), (\ell_\Diamond, \{2\}, 2), (\ell_\Diamond, \{3\}, 2), (b_1, \{0\}, 1)\},\\
W^2_{post} &:= \{(\ell_\Box,]3, +\infty[, 1), (\ell_\Diamond, \{0\}, 1), (\ell_\Diamond, \{3\}, 1), (b_1, \{0\}, 1)\},\\
W^3_{post} &:= \{(\ell_\Box,]3, +\infty[, 1), (\ell_\Diamond, \{2\}, 1), (\ell_\Diamond, \{3\}, 1), (b_0, \{3\}, 1)\},\\
W^4_{post} &:= \{(\ell_\Box,]3, +\infty[, 1), (b_0, \{3\}, 1)\}.
\end{aligned}$$

A similar procedure enables to compute the others elements of $Post(W)$ from W^1, W^2 and W^3.

The two following definitions present some notations and vocabulary used in the presentation of our algorithm.

Definition 4.33. *We note $H_0 := H(s_0)$ the word of $\mathcal{H}$ corresponding to the initial state of $\mathcal{S}_{\mathcal{B},\neg\Phi}$.*

Definition 4.34. *We say that a word $W \in \mathcal{H}$ is* accepting *iff all the locations (of $\mathcal{A}_{\neg\Phi}$ or $\mathcal{B}$) present in the first components of the 3-tuples it contains are accepting (such words correspond to accepting states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$).*
We note $\mathcal{F} \subseteq \mathcal{H}$ the set of accepting words of $\mathcal{H}$.

Algorithm 3 presents an (classical) algorithm for the model-checking of MITL in which the reachable words of $\mathcal{H} = \mathcal{S}_{\mathcal{B},\neg\Phi}/\equiv$ are computed 'on the fly'. It is a classical algorithm to explore the (finite) state space of $\mathcal{H}$. It works 'on the fly' stopping as soon as an accepting state (i.e. word) of $\mathcal{H}$ is reached. Indeed, reaching an accepting state of $\mathcal{H}$ means there is a path from the initial state to an accepting state in $\mathcal{H}$, so that $\mathcal{B} \not\models \Phi$. If there exists no such state, the algorithm ends when all the states of $\mathcal{H}$ were explored, answering that $\mathcal{B} \models \Phi$. Depending on the data structure used to represent the set *ToExplore*, this algorithm uses a classical breadth-first or depth-first graph traversal. Remark that the frontier *ToExplore* is maintained in an efficient manner. Indeed, the Algorithm 4 would give the same result as Algorithm 3.

Algorithm 3 MITLModelChecking

Input: A TA $\mathcal{B}$ and the ATA $\mathcal{A}_{\neg\Phi}$, for $\Phi \in$ MITL.
Output: 'true' iff $\mathcal{B} \models \Phi$.

```
ToExplore ← H_0
Explored ← ∅
while ToExplore ≠ ∅ do
    Remove some element W from ToExplore
    if W is accepting then
        return 'false'
    end if
    Explored = Explored ∪ {W}
    ToExplore = ToExplore ∪ (Post(W) \ Explored)
end while
return 'true'
```

Algorithm 4 NaiveMITLModelChecking

Input: A TA $\mathcal{B}$ and the ATA $\mathcal{A}_{\neg\Phi}$, for $\Phi \in$ MITL.

Output: 'true' iff $\mathcal{B} \models \Phi$.

```
1: S ← H_0
2: S_pre ← ∅
3: while S ≠ S_pre do
4:     if there is an accepting W ∈ S then
5:         return 'false'
6:     end if
7:     S_pre ← S
8:     S ← S ∪ Post(S)
9: end while
10: return 'true'
```

4.3 Antichain-based heuristic

The aim of this section is to elaborate a heuristic to improve the practical efficiency of Algorithm 3. Many recent works praise the improvement of algorithms thanks to the use of *antichains* [18, 30, 64]. In particular, efficient algorithms using antichains have been produced for the LTL satisfiability and model-checking [66], as well as for the LTL reactive synthesis problem [32]. We follow the footsteps of these authors showing how antichains can be used to improve the Algorithm 3 of the previous section.

We start by formally defining the notion of *antichain*, based on the notion of *partial order*.

Definition 4.35. *A* partial order *over a finite set S is a binary relation $\preceq \subseteq S \times S$ which is reflexive, transitive and antisymmetric. For $s_1, s_2 \in S$, if $s_1 \preceq s_2$, we say that s_1 is smaller than s_2 (or s_2 is greater than s_1). If $s_1 \npreceq s_2$ and $s_2 \npreceq s_1$, we say that s_1 and s_2 are incomparable.*

Definition 4.36. *Let S be a finite set and $\preceq \subseteq S \times S$ be a partial order. An*

antichain over S is a set of pairwise incomparable elements.

For instance, when considering a partial order over a finite set S, the set of its minimal elements is an antichain over S, as well as the set of its maximal elements.

Definition 4.37. *Let S be a finite set and $\preceq \subseteq S \times S$ be a partial order. We define the set of the minimal elements of S by:*

$$\min(S) = \{s \in S \mid \forall s' \in S,\ (s' \preceq s \Rightarrow s' = s)\}.$$

Similarly, we define the set of the maximal elements of S by:

$$\max(S) = \{s \in S \mid \forall s' \in S,\ (s \preceq s' \Rightarrow s' = s)\}.$$

For the rest of this section, we still fix an MITL formula Φ and assume that the OCATA representing the negation of Φ is $\mathcal{A}_{\neg\Phi} = (\Sigma, L, \ell_0, F, \delta)$. We also fix a TA $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$ and the timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi} = (\Sigma, S, s_0, \rightsquigarrow, \rightarrow, \alpha)$ (see Definition 4.17). We also observe $\mathcal{H} = \mathcal{S}_{\mathcal{B},\neg\Phi}/\equiv$ (see Definition 4.27), whose elements are words over the alphabet

$$\Lambda = (B \cup L) \times REG \times \{1, 2, \ldots, \max(\max_{\ell \in L}(\max_\ell), n)\},$$

where $\max_\ell$ is the maximal number of intervals that can be present in location $\ell \in L$ of $\mathcal{A}_{\neg\Phi}$, and $n := |X|$ is the number of clocks of $\mathcal{B}$.

In the sequel, we define an order on the elements of S and show how we can exploit it to improve the Algorithm 3 of the previous section. Let $s, s' \in S$ such that $s \subseteq s'$ (in particular, they contain the same configuration of $\mathcal{B}$). If we have no accepting run from s', it is not interesting to observe it in the algorithm. If we have an accepting run from s', we must have an accepting run from s (see the definition of the transition function of OCATAs). So, in Algorithm 3, it is not necessary to observe s' if we have already observed s because the first accepting

run we will find will suffice to prove that $\mathcal{B} \not\models \Phi$. In fact, we can extend this remark to configurations whose clock values are in the same regions and whose order of the fractional parts (of their clock values) is the same. We will define a partial order $\sqsubseteq$ on the words of $\mathcal{H}$ that enable to verify such an 'inclusion' on the sets of configurations they give. Our improved algorithm will then consists in only observing the antichain of the minimal elements of $\mathcal{H}$ for this partial order.

Definition 4.38. *We define the partial order $\sqsubseteq$ on $\mathcal{H}$ as follows. Let W^1, $W^2 \in \mathcal{H}$ and suppose that $W^1 = w_1^1 w_2^1 \dots w_n^1$ and $W^2 = w_1^2 w_2^2 \dots w_m^2$. $W^1 \sqsubseteq W^2$ iff:*

- *$n \leqslant m$, and*
- *there exists a strictly increasing function $f : \{1, 2, \dots, n\} \longrightarrow \{1, 2, \dots, m\}$ such that:*
  1. *$\forall 1 \leqslant i \leqslant n, \forall \ell \in B \ : \ (\ell, r, k) \in w_i^1 \ \Rightarrow \ (\ell, r, k) \in w_{f(i)}^2$, and*
  2. *$\forall \ell \in L$, there exists a strictly increasing function $f_\ell : \mathbb{N} \longrightarrow \mathbb{N}$ such that: $\forall 1 \leqslant i \leqslant n, \ (\ell, r, k) \in w_i^1 \ \Rightarrow \ (\ell, r, f_\ell(k)) \in w_{f(i)}^2$.*

Remark 4.39. Remark that we asked the existence of strictly increasing functions f_ℓ, *for all $\ell \in \mathcal{A}$*. In fact, only functions f_ℓ for an $\ell \in L$ *present in an element of a letter of* W^1 are relevant. Moreover, those functions f_ℓ are defined from $\mathbb{N}$ to $\mathbb{N}$. In fact, we only need each f_ℓ to be defined from $\{1, 2, \dots, \max_\ell\}$ (where $\max_\ell$ is the maximal number of intervals that can be present in location ℓ) to itself, as there will never be more than those number of intervals associated to location ℓ.

Remark 4.40. $H(s^1) = W^1 \sqsubseteq W^2 = H(s^2)$ **iff** there exists $s_0^2 \subseteq s^2$ such that $H(s^1) = H(s_0^2)$, i.e. (Proposition 4.26) $s^1 \equiv s_0^2$. (s_0^2 can be found only keeping the 3-tuples of W^2 matched with a trio of W^1 thanks to the functions f and f_ℓ.)

Example 4.41. Let $\ell^{\mathcal{B}}$ be a location of $\mathcal{B}$ and ℓ be a location of $\mathcal{A}_{\neg\Phi}$. Suppose that $\mathcal{B}$ has 1 clock and $c_{max} = 3$. Let us note:

$$\begin{aligned} W &:= \{(\ell, \{0\}, 1), (\ell, \{3\}, 2)\} \ \{(\ell,]0, 1[, 1)\} \ \{(\ell,]2, 3[, 2), (\ell^{\mathcal{B}},]1, 2[, 1)\}, \text{ and} \\ W' &:= \{(\ell, \{0\}, 1), (\ell, \{3\}, 3)\} \ \{(\ell,]0, 1[, 1)\} \ \{(\ell,]2, 3[, 2)\} \\ &\quad \{(\ell,]0, 1[, 2), (\ell,]2, 3[, 3), (\ell^{\mathcal{B}},]1, 2[, 1)\}. \end{aligned}$$

We have that $W \sqsubseteq W'$, considering:

- $f : \{1,2,3\} \longrightarrow \{1,2,3,4\}$ defined by $f(1) = 1$, $f(2) = 2$, $f(3) = 4$;
- $f_{\ell^2} : \mathbb{N} \longrightarrow \mathbb{N}$ defined by $f(1) = 1$, $f(2) = 3$ and $\forall n \in \mathbb{N}\backslash\{1,2\}, f(n) = n+1$;

Let us now give an example to support Remark 4.40. For instance, $W = H(s^1)$ and $W' = H(s^2)$, for:

$$\begin{array}{rcl} s^1 & := & \{(\ell,[0,0.2]),(\ell,[2.4,3]),(\ell^{\mathcal{B}},1.4)\}, \text{ and} \\ s^2 & := & \{(\ell,[0,0.4]),(\ell,[0.9,2.5]),(\ell,[2.9,3]),(\ell^{\mathcal{B}},1.9)\} \end{array}$$

Considering

$$\begin{array}{rcl} s_0^2 & := & \{(\ell,[0,0.4]),(\ell,[2.9,3]),(\ell^{\mathcal{B}},1.9)\}, \text{ we have} \\ H(s_0^2) & := & \{(\ell,\{0\},1),(\ell,\{3\},2)\}\ \{(\ell,]0,1[,1)\}\{(\ell,]2,3[,2),(\ell^{\mathcal{B}},]1,2[,1)\}, \end{array}$$

and hence $H(s_0^2) = H(s^1)$ $(= W)$.

Our following aim is to prove that $\sqsubseteq$ induces a *forward simulation* on the words of $\mathcal{H}$. We prove this claim after having formally defined the notion of forward simulation, concerning the transitions $\xrightarrow{\sigma}$ and $\longrightarrow_T$ defined on the words of $\mathcal{H}$.

Definition 4.42. *Let $\preccurlyeq$ be an order on the word of $\mathcal{H}$. We say that $\preccurlyeq$ induces a forward simulation on $\mathcal{H}$ iff:*

$$\begin{array}{ll} \textit{for all } W,W',V' \textit{ words of } \mathcal{H} \textit{ and } \sigma \in \Sigma \textit{ such that} & W \preccurlyeq W' \textit{ and } W' \xrightarrow{\sigma} V', \\ \textit{there exists } V \textit{ a word of } \mathcal{H} \textit{ such that} & V \preccurlyeq V' \textit{ and } W \xrightarrow{\sigma} V; \end{array}$$

and

$$\begin{array}{ll} \textit{for all } W,W',V' \textit{ words of } \mathcal{H} \textit{ and } \sigma \in \Sigma \textit{ such that} & W \preccurlyeq W' \textit{ and } W' \longrightarrow_T V', \\ \textit{there exists } V \textit{ a word of } \mathcal{H} \textit{ such that} & V \preccurlyeq V' \textit{ and } W \longrightarrow_T V. \end{array}$$

Proposition 4.43. $\sqsubseteq$ induces a forward simulation on the words of $\mathcal{H}$.

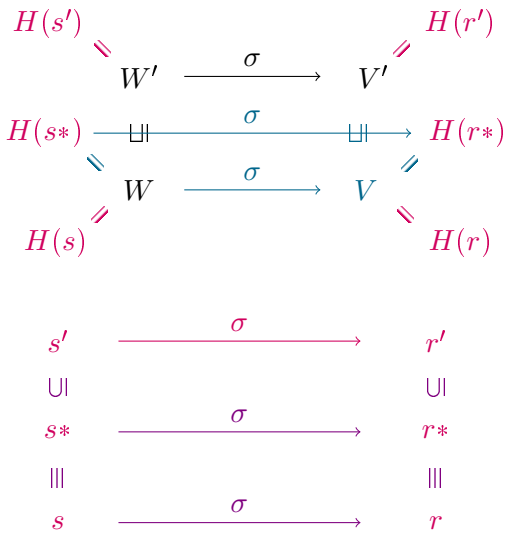


Figure 4.10: (above) Summarize on $\mathcal{H}$. (below) Summarize on configurations.

Proof. <u>*Discrete transition:*</u> Let W, W', V' be three words of $\mathcal{H}$ and $\sigma \in \Sigma$ such that $W \sqsubseteq W'$ and $W' \xrightarrow{\sigma} V'$. We must prove that there exists a word $V \in \mathcal{H}$ such that $W \xrightarrow{\sigma} V$ and $V \sqsubseteq V'$.
Figure 4.3 summarizes the situation and the notations.
Let us suppose that $W = H(s)$ and $W' = H(s')$ for certain $s, s' \in S$. Thanks to Remark 4.40, we know that there exists $s* \subseteq s'$ such that $H(s) = H(s*)$. As $W' \xrightarrow{\sigma} V'$, there exists $r' \in H^{-1}(V')$ such that $s' \xrightarrow{\sigma} r'$. As $s* \subseteq s'$ and by definition of $\xrightarrow{\sigma}$ on S, there is a configuration $r* \subseteq r'$ such that $s* \xrightarrow{\sigma} r*$ and so $H(s*) \xrightarrow{\sigma} H(r*)$. Now, as $H(s) = H(s*)$ (i.e. $s \equiv s*$ by Proposition 4.26) and $s* \xrightarrow{\sigma} r*$, by Proposition 4.23, there exists $r \in S$ such that $s \xrightarrow{\sigma} r$ (and so $W = H(s) \xrightarrow{\sigma} H(r)$) and $r \equiv r*$. Now, as $r \equiv r* \subseteq r'$, by Remark 4.40, $H(r) \sqsubseteq H(r') = V'$: $H(r)$ is the V we were looking for.
<u>*Timed transition:*</u> We can conclude thanks to a proof similar to that of the 'discrete transition' case, thanks to Remark 4.40, and Propositions 4.23 and 4.26. $\square$

Thanks to the forward simulation induced by $\sqsubseteq$, we are now able to set up an algorithm to solve the MITL model-checking problem. We recall that we are

looking for a path from the initial word of $\mathcal{H}$, H_0, to an accepting word of $\mathcal{H}$: $\mathcal{B} \models \Phi$ iff there is no such path. Our first aim is so to compute the reachable words of $\mathcal{H}$: in the following, we elaborate algorithms to compute them. When we will have obtained an *efficient* algorithm for this (thanks to the forward simulation induced by $\sqsubseteq$), we will elaborate a complete algorithm to perform *on the fly* MITL model-checking.

In following lemmas, we show that, in way to verify that $\mathcal{B} \models \Phi$, it is equivalent to compute:

1. the reachable words of $\mathcal{H}$,
2. the *upward closure* of the reachable words of $\mathcal{H}$,
3. the antichain of the *minimal elements* of the reachable words of $\mathcal{H}$.

We present these lemmas after having defined what is the *upward closure* of an ordered set and fix notations for the reachable and accepting words of $\mathcal{H}$.

Definition 4.44. *Let E be a set and $\leqslant$ a partial order on the elements of E. Let $X \subseteq E$. We note $\uparrow X$ the* upward closure *of X defined as follows:*

$$\uparrow X \quad := \{e \in E \mid \exists x \in X \text{ such that } x \leqslant e\}.$$

Definition 4.45. *We note $Reach(\mathcal{H})$ the reachable words of $\mathcal{H}$. We note $Accept(\mathcal{H})$ the accepting words of $\mathcal{H}$.*

Thanks to these notations, we can rewrite the aim of an MITL model-checking algorithm has follows:

$$\mathcal{B} \models \Phi \quad \text{iff} \quad Reach(\mathcal{H}) \cap Accept(\mathcal{H}) = \varnothing.$$

In way to prove the desired lemmas, we first define what is a *downward closed* set, and prove that $Accept(\mathcal{H})$ owns this property.

Definition 4.46. *Let E be a set and $\leqslant$ a partial order on the elements of E. Let $X \subseteq E$. We say that X is* downward closed *if:*

$$\forall x \in X,\ \forall y \in E\colon\ (y \leqslant x) \Rightarrow (y \in X).$$

Lemma 4.47. *$Accept(\mathcal{H})$ is downward closed.*

Proof. Let $W \in Accept(\mathcal{H})$ and $W' \in \mathcal{H}$ such that $W' \sqsubseteq W$. We must show that $W' \in Accept(\mathcal{H})$. Indeed, as $W' \sqsubseteq W$, each 3-tuple of W' as a corresponding $3-tuple$ in W carrying the same location of $\mathcal{A}_{\neg\Phi}$ or $\mathcal{B}$. Now, as $W \in Accept(\mathcal{H})$, all the locations present in each of its 3-tuples is accepting, so that this is also the case for W'. This proves that $W' \in Accept(\mathcal{H})$. □

The following lemmas once again rewrite, in two new manners, the aim of an MITL model-checking algorithm.

Lemma 4.48.

$$Reach(\mathcal{H}) \cap Accept(\mathcal{H}) = \varnothing \qquad \textit{iff} \qquad \uparrow Reach(\mathcal{H}) \cap Accept(\mathcal{H}) = \varnothing.$$

Proof. ($\Rightarrow$) Let $x \in \uparrow Reach(\mathcal{H})$. There is $y \in Reach(\mathcal{H})$ such that $y \sqsubseteq x$. Let us suppose by contradiction that $x \in Accept(\mathcal{H})$. Then, as $Accept(\mathcal{H})$ is downward-closed and $y \sqsubseteq x$, $y \in Accept(\mathcal{H})$ what is impossible as, by hypothesis, $Reach(\mathcal{H}) \cap Accept(\mathcal{H}) = \varnothing$.
($\Leftarrow$) Let $x \in Reach(\mathcal{H})$, we must show $x \notin Accept(\mathcal{H})$. Indeed, as $x \in Reach(\mathcal{H})$, in particular $x \in \uparrow Reach(\mathcal{H})$ and so $x \notin Accept(\mathcal{H})$ because, by hypothesis, $\uparrow Reach(\mathcal{H}) \cap Accept(\mathcal{H}) = \varnothing$. □

Lemma 4.49.

$$\uparrow Reach(\mathcal{H}) \cap Accept(\mathcal{H}) = \varnothing \qquad \textit{iff} \qquad \min(Reach(\mathcal{H})) \cap Accept(\mathcal{H}) = \varnothing.$$

Proof. ($\Rightarrow$) Let us suppose that $\uparrow Reach(\mathcal{H}) \cap Accept(\mathcal{H}) = \varnothing$. Let $x \in \min(Reach(\mathcal{H}))$. We suppose by contradiction that $x \in Accept(\mathcal{H})$. In particular $x \in \uparrow Reach(\mathcal{H})$, which contradicts our hypothesis.

($\Leftarrow$) Let us suppose that $\min(\mathit{Reach}(\mathcal{H})) \cap \mathit{Accept}(\mathcal{H}) = \emptyset$. Let $x \in \uparrow \mathit{Reach}(\mathcal{H})$. We suppose by contradiction that $x \in \mathit{Accept}(\mathcal{H})$. As $x \in \uparrow \mathit{Reach}(\mathcal{H})$, there exists $x' \in \min(\mathit{Reach}(\mathcal{H}))$ such that $x' \sqsubseteq x$. Then, as $x \in \mathit{Accept}(\mathcal{H})$ and $\mathit{Accept}(\mathcal{H})$ is downward closed, $x' \in \mathit{Accept}(\mathcal{H})$: this contradicts our hypothesis. $\square$

We start by presenting Algorithm 5. It is a *theoretical* algorithm that enables to compute $\uparrow(\mathit{Reach}(\mathcal{H}))$. We prove that this algorithm is correct. Nevertheless, this *theoretical* algorithm cannot be really implemented because $\uparrow(\mathit{Reach}(\mathcal{H}))$ is an infinite set. Fortunately, $\uparrow(\mathit{Reach}(\mathcal{H}))$ can easily be represented by the antichain of its minimal elements. In the sequel, we will so present an effective algorithm to compute $\min(\uparrow(\mathit{Reach}(\mathcal{H})))$.

Algorithm 5 UpwardClosureReach

Input: H_0

Output: $\uparrow \mathit{Reach}(\mathcal{H})$.

```
1: R_pre ← ∅
2: R ← ↑(H_0)
3: S ← ↑(H_0)
4: while R_pre ≠ R do
5:     S ← ↑Post(S)
6:     R_pre ← R
7:     R ← R ∪ S
8: end while
9: return R
```

The proof of correctness of this algorithm is based on the two following lemmas.

Lemma 4.50. *Let $A \subseteq \mathcal{H}$. Then:*

$$\uparrow \mathit{Post}(\uparrow A) = \uparrow \mathit{Post}(A).$$

Proof. ($\subseteq$) Let $W \in \uparrow \mathit{Post}(\uparrow A)$, we must prove that $W \in \uparrow \mathit{Post}(A)$. As $W \in$

$\uparrow Post(\uparrow A)$, there exists $W_1 \in Post(\uparrow A)$ such that $W_1 \sqsubseteq W$. Moreover, there exists $W_{pre} \in \uparrow A$, $\sigma \in \Sigma$ and $W_p \in \mathcal{H}$ such that:

$$W_{pre} \longrightarrow_T W_p \xrightarrow{\sigma} W_1.$$

As $W_{pre} \in \uparrow A$, there exists $W'_{pre} \in A$ such that $W'_{pre} \sqsubseteq W_{pre}$. By Proposition 4.26 and Proposition 4.23, there exists W'_p and $W_2 \in \mathcal{H}$ such that:

$$W'_{pre} \longrightarrow_T W'_p \xrightarrow{\sigma} W_2 \text{ and } W_2 \sqsubseteq W_1.$$

So, $W_2 \in Post(W'_{pre})$ and, as $W'_{pre} \in A$, we have: $W_2 \in Post(A)$. Hence, as $W_2 \sqsubseteq W_1 \sqsubseteq W$, we have that $W \in \uparrow Post(A)$.
($\supseteq$) We must show that $\uparrow Post(A) \subseteq \uparrow Post(\uparrow A)$. Indeed, we have $A \subseteq \uparrow A$, so that $Post(A) \subseteq Post(\uparrow A)$ (it is not difficult to see that function $Post$ is monotonic), and hence $\uparrow Post(A) \subseteq \uparrow Post(\uparrow A)$. □

Lemma 4.51. *Let $n \in \mathbb{N}$ and $(E_i)_{i=1}^n$ a family of sets. Then:*

$$\bigcup_{i=1}^n \uparrow (E_i) = \uparrow \bigcup_{i=1}^n E_i.$$

Proof. We have:

$$\begin{array}{ll} & x \in \bigcup_{i=1}^n \uparrow (E_i) \\ \text{iff} & x \in \uparrow (E_1) \text{ or } x \in \uparrow (E_2) \text{ or } \ldots \text{ or } x \in \uparrow (E_n) \\ \text{iff} & \exists y \text{ in } E_1 \text{ or } E_2 \text{ or } \ldots \text{ or } E_n \text{ such that } y \sqsubseteq x \\ \text{iff} & \exists y \in \bigcup_{i=1}^n E_i \text{ such that } y \sqsubseteq x \\ \text{iff} & x \in \uparrow \bigcup_{i=1}^n E_i. \end{array}$$

□

We can now prove the correctness of Algorithm 5.

Theorem 4.52. *Algorithm 5 returns $\uparrow Reach(\mathcal{H})$.*

Proof. We first remark that this algorithm terminates because $\mathcal{H}$ is finite. Let us note k the number of passing in the *while* loop performed before Algorithm 5 terminates. We also note R^{end} and R^{end}_{pre} the respective values of variables R and R_{pre} after the *while* loop. We remark that $R^{end} = \bigcup_{i=0}^{k} (\uparrow Post)^i(\uparrow H_0)$. We must show that:

$$\bigcup_{i=0}^{k} (\uparrow Post)^i(\uparrow H_0) = \uparrow Reach(\mathcal{H}).$$

($\subseteq$) On the one hand, to recursively use Lemma 4.50 gives the following equalities, for all $n \in \mathbb{N}$:

$$\begin{array}{rl} & (\uparrow Post)^n(\uparrow H_0) \\ := & (\uparrow Post)^{n-1}(\uparrow Post(\uparrow H_0)) \\ = & (\uparrow Post)^{n-2}(\uparrow Post(\uparrow Post(H_0))) \\ = & (\uparrow Post)^{n-3}(\uparrow Post(\uparrow Post(Post(H_0)))) \\ = & ... \\ = & \uparrow Post(\uparrow Post(Post^{n-2}(H_0))) \\ = & \uparrow Post^n(H_0). \end{array}$$

We so have:

$$\begin{array}{rl} & \bigcup_{i=0}^{k} (\uparrow Post)^i(\uparrow H_0) \\ = & \bigcup_{i=0}^{k} \uparrow (Post^i(H_0)) \\ = & \uparrow (\bigcup_{i=0}^{k} Post^i(H_0)) \qquad \text{(by Lemma 4.51)} \\ \subseteq & \uparrow (\bigcup_{n\in\mathbb{N}} Post^n(H_0)) \\ = & \uparrow Reach(\mathcal{H}). \end{array}$$

($\supseteq$) We first claim that:

$$\uparrow Post(R^{end}) \subseteq R^{end}. \tag{4.6}$$

To prove this, we note S^i the value of variable S after the ith passing in the *while* loop. We have: $S^0 = \uparrow (H_0)$ and $S^{i+1} = \uparrow Post(S^i)$ for all $i \geqslant 0$; $R^{end} = \bigcup_{i=0}^{k} S^i$ and $R^{end}_{pre} = \bigcup_{i=0}^{k-1} S^i$. Moreover, as the algorithm only performed k passing in the loop, we have that: $R^{end}_{pre} = R^{end}$, i.e. $\bigcup_{i=0}^{k-1} S^i = \bigcup_{i=0}^{k} S^i$, and, in particular, $S^k \subseteq \bigcup_{i=0}^{k-1} S^i$.

Let $x \in \uparrow Post(R^{end})$. In way to prove 4.6, we will show that $x \in R^{end}$.

$$\begin{array}{rcll} x & \in & \uparrow Post(R^{end}) & \\ & = & \uparrow Post(\bigcup_{i=0}^{k} S^i) & \\ & = & \uparrow \bigcup_{i=0}^{k} Post(S^i) & \text{(by definition of } Post) \\ & = & \bigcup_{i=0}^{k} \uparrow Post(S^i) & \text{(by Lemma 4.51)} \end{array}$$

We distinguish two cases:

(i) If $x \in \uparrow Post(S^i)$, for $0 \leqslant i < k$, then, $x \in S^{i+1}$ (by definition of S^{i+1}) and so $x \in R^{end}$.

(ii) Else, $x \in \uparrow Post(S^k)$. Then, as $S^k \subseteq \bigcup_{i=0}^{k-1} S^i$, $x \in \uparrow Post(\bigcup_{i=0}^{k-1} S^i) = \bigcup_{i=0}^{k-1} \uparrow Post(S^i)$. We can conclude that $x \in R^{end}$ thanks to case (i).

We are now able to prove that $R^{end} \supseteq \uparrow Reach(\mathcal{H})$.
Let $W \in \uparrow Reach(\mathcal{H}) = \uparrow (\bigcup_{n \in \mathbb{N}} Post^n(H_0))$. We suppose by contradiction that $W \notin R^{end}$. As $R^{end} = \uparrow (\bigcup_{i=0}^{k} Post^i(H_0))$, it is only possible if there is a path

$$\begin{array}{l} W_0 \longrightarrow_T W'_0 \xrightarrow{\sigma_1} W_1 \longrightarrow_T W'_1 \xrightarrow{\sigma_2} W_2 \ldots \xrightarrow{\sigma_k} W_k \\ \quad \longrightarrow_T W'_k \xrightarrow{\sigma_{k+1}} W_{k+1} \ldots \xrightarrow{\sigma_{k+1}} W_{k+l} \end{array}$$

in $\mathcal{H}$, for a certain $l \geqslant 1$, with $W_0 = H_0$ and $W_{k+l} \sqsubseteq W$. By definition of R^{end}, we have that, for $0 \leqslant i \leqslant k$: $W_i \in R^{end}$ and $W'_i \in R^{end}$. Moreover, as $W_k \in R^{end}$ and $\uparrow Post(R^{end}) \subseteq R^{end}$, we have that $W_{k+1} \in R^{end}$. Following the same reasoning, we recursively obtain that $W_{k+2} \in R^{end}$, $W_{k+3} \in R^{end}$, $\ldots$, $W_{k+l} \in R^{end}$. Now, we know that $R^{end} = \uparrow (\bigcup_{i=0}^{k} Post^i(H_0))$, $W_{k+l} \in R^{end}$ and $W_{k+l} \sqsubseteq W$, so that $W \in R^{end}$. $\square$

While Algorithm 5 is correct, it is unexploitable in practice because it manipulates potentially infinite objects ($\uparrow Post^i(\uparrow H_0), \ldots$). To overcome this problem, we present Algorithm 6. Based on the results of Lemmas 4.48 and 4.49, the aim of this algorithm is to only compute the antichain of the minimal elements

Algorithm 6 MinReach

Input: H_0

Output: $\min(Reach(\mathcal{H}))$.

1: $\tilde{R}_{pre} \leftarrow \emptyset$
2: $\tilde{R} \leftarrow \min(H_0)$
3: $\tilde{S} \leftarrow \min(H_0)$
4: **while** $\tilde{R}_{pre} \neq \tilde{R}$ **do**
5: $\quad \tilde{S} \leftarrow \min(Post(\tilde{S}))$
6: $\quad \tilde{R}_{pre} \leftarrow \tilde{R}$
7: $\quad \tilde{R} \leftarrow \min(\tilde{R} \cup \tilde{S})$
8: **end while**
9: **return** $\tilde{R}$

of $Reach(\mathcal{H})$. To do this, it proceeds as Algorithm 5 but only maintains the minimal elements of $Post(S)$, which finitely represent $\uparrow Post(S)$.

The proof of correctness of Algorithm 6 is based on the two following lemmas.

Lemma 4.53. *Let $E \subseteq \mathcal{H}$. We have that:*

$$\min(Post(\min(E))) = \min(Post(E)).$$

Proof. ($\subseteq$) Let $x \in \min(Post(\min(E)))$, i.e. $x \in Post(\min(E))$ and for all $y \in Post(\min(E))$, $y \not\sqsubseteq x$. In particular, $x \in Post(E)$. We must still show that, for all $y \in Post(E)$, $y \not\sqsubseteq x$. Let $y \in Post(E)$: there exists $\sigma \in \Sigma$, $z, z' \in E$ and $z_m \in \min(E)$ such that:

$$z \longrightarrow_T z' \xrightarrow{\sigma} y \text{ and } z_m \sqsubseteq z.$$

By Propositions 4.26 and 4.23, there exists $z'_m, y_m \in \mathcal{H}$ such that:

$$z_m \longrightarrow_T z'_m \xrightarrow{\sigma} y_m \text{ and } y_m \sqsubseteq y.$$

So, $y_m \in Post(z_m)$ and, as $z_m \in \min(E)$, we have: $y_m \in Post(\min(E))$, so that $y_m \not\sqsubseteq x$. As $y_m \sqsubseteq y$, we also have that $y \not\sqsubseteq x$ (else, we would have that

$y_m \sqsubseteq y \sqsubseteq x$ and so $y_m \sqsubseteq x$).
($\supseteq$) Let $x \in \min(Post(E))$, i.e. $x \in Post(E)$ and for all $y \in Post(E)$, $y \not\sqsubseteq x$. Then, there exists $\sigma \in \Sigma$, $z, z' \in E$ and $z_m \in \min(E)$ such that:

$$z \longrightarrow_T z' \xrightarrow{\sigma} x \text{ and } z_m \sqsubseteq z.$$

By Propositions 4.26 and 4.23, there exists $z'_m, x_m \in \mathcal{H}$ such that:

$$z_m \longrightarrow_T z'_m \xrightarrow{\sigma} x_m \text{ and } x_m \sqsubseteq x.$$

So, $x_m \in Post(z_m)$ and, as $z_m \in \min(E)$, we have: $x_m \in Post(\min(E))$. In particular, $x_m \in Post(E)$ and so, $x_m \sqsubseteq x$ implies that $x_m = x$. Hence, $x \in Post(\min(E))$. As we know that, for all $y \in Post(E)$, $y \not\sqsubseteq x$, in particular, for all $y \in Post(\min(E))$, $y \not\sqsubseteq x$ and so: $x \in \min(Post(\min(E)))$. □

Lemma 4.54. *Let $A, B \subseteq \mathcal{H}$. We have that:*

$$\min(\min(A) \cup \min(B)) = \min(A \cup B).$$

Proof. ($\subseteq$) Let $x \in \min(\min(A) \cup \min(B))$. In particular, $x \in A \cup B$ and for all $y \in \min(A) \cup \min(B)$, $y \not\sqsubseteq x$. It remains to prove that for all $y \in A \cup B$, $y \not\sqsubseteq x$. Let $y \in A \cup B$. Let us suppose that $y \in A$, the same reasoning holds if $y \in B$. Then, there exists $y_m \in \min(A)$ such that $y_m \sqsubseteq y$. But we know that for such an $y_m \in \min(A)$, we have that $y_m \not\sqsubseteq x$. Then, $y_m \sqsubseteq y$ induces that $y \not\sqsubseteq x$.
($\supseteq$) Let $x \in \min(A \cup B)$, then $x \in A \cup B$ and, for all $y \in A \cup B$, we have that $y \not\sqsubseteq x$. Let us suppose that $x \in A$, the same reasoning holds if $x \in B$. In particular, for all $a \in A$, $a \not\sqsubseteq x$, and so $x \in \min(A)$. Hence, as $\min(A) \subseteq \min(A) \cup \min(B)$, we have that $x \in \min(A) \cup \min(B)$. It remains to prove that, for all $y \in \min(A) \cup \min(B)$, $y \not\sqsubseteq x$: it is clear as we know that, for all $y \in A \cup B$, $y \not\sqsubseteq x$. □

The following theorem states that Algorithm 6 is correct.

Theorem 4.55. *Algorithm 6 returns* $\min(Reach(\mathcal{H}))$.

Proof. To prove this theorem, we will show that, at each passing in the *while* loop, Algorithm 6 stores in its variables $\tilde{S}$ et $\tilde{R}$ the respective minimal elements of what contains variables S and R of Algorithm 5, at the same passing in its *while* loop. To do this, let us note S^i, R^i, $\tilde{S}^i$ and $\tilde{R}^i$ the respective contents of variables S, R, $\tilde{S}$ and $\tilde{R}$ at the end of the ith passing in the *while* loop, for $i \geqslant 0$. We must show that, $\forall i \geqslant 0$, $\tilde{S}^i = \min(S^i)$ and $\tilde{R}^i = \min(R^i)$.
Before to enter the loop: we have $S^0 = \uparrow (H_0)$ and $\tilde{S}^0 = \min(H_0) = \min(S^0)$. Moreover, $R^0 = \uparrow (H_0)$ and $\tilde{R}^0 = \min(H_0) = \min(R^0)$, what we wanted.
In the loop: suppose that, $\forall 0 \leqslant i \leqslant j$, after the ith passing in the loop, we have $\tilde{S}^i = \min(S^i)$ and $\tilde{R}^i = \min(R^i)$. We now show that $\tilde{S}^{j+1} = \min(S^{j+1})$ and $\tilde{R}^{j+1} = \min(R^{j+1})$, after the $(j+1)$th passing in the loop.
After this $(j+1)$th passing, we have:

$$\begin{aligned}
\tilde{S}^{j+1} &= \min(Post(\tilde{S}^j)) \\
&= \min(Post(\min(S^j)) && \text{(by induction hypothesis)} \\
&= \min(Post(S^j)) && \text{(by Lemma 4.53)} \\
&= \min(\uparrow Post(S^j)) && \text{(by definition of } \uparrow \text{ and min)} \\
&= \min(S^{j+1}) && \text{(see Algorithm 5).}
\end{aligned}$$

Moreover:

$$\begin{aligned}
\tilde{R}^{j+1} &= \min(\tilde{R}^j \cup \tilde{S}^{j+1}) \\
&= \min(\min(R^j) \cup \min(S^{j+1})) && \text{(by induction hypothesis} \\
& && \text{and because } \tilde{S}^{j+1} = \min(S^{j+1}))) \\
&= \min(R^j \cup S^{j+1}) && \text{(by Lemma 4.54)} \\
&= \min(R^{j+1}) && \text{(see Algorithm 5).}
\end{aligned}$$

So, we have that $\forall i \geqslant 0$, $\tilde{S}^i = \min(S^i)$ and $\tilde{R}^i = \min(R^i)$. As we know Algorithm 5 always stops, it is also always the case for Algorithm 6. Indeed, suppose that Algorithm 5 stops after k passing in the *while*, it means $R^{k-1} = R^k$ (observing the content of R_{pre} at each passing in the *while* loop). But $R^{k-1} = R^k$ iff $\min(R^{k-1}) = \min(R^k)$ iff $\tilde{R}^{k-1} = \tilde{R}^k$, which is the condition thanks to which stops the *while* loop of Algorithm 6. Moreover, as we proved that, if Algorithm 5 stops after k passing in the *while* loop, it returns $R^k = \uparrow Reach(\mathcal{H})$,

we have that Algorithm 6 indeed returns $\tilde{R}^k = \min(R^k) = \min(\uparrow Reach(\mathcal{H})) = \min(Reach(\mathcal{H}))$, what we wanted. $\square$

Algorithm 6 can still be improved. Algorithm 7 will only apply operator *Post* on the elements x such that *Post* has not been applied on an element $y \sqsubseteq x$ yet. In the sequel, we will show that, when computing the *Post* of such an element x, the obtained elements are not minimal and are so useless.
Algorithm 7 uses the operator *minRel* defined as follows:

Definition 4.56. *Let $S, R \subseteq \mathcal{H}$. We define:*

$$minRel(S, R) := \{s \in S \mid \forall s' \in S,\ s' \not\sqsubseteq s \text{ and } \forall r \in R,\ r \not\sqsubseteq s\}.$$

Algorithm 7 OptimizedMinReach

Input: H_0

Output: $\min(Reach(\mathcal{H}))$.

```
R*_pre ← ∅
R* ← min(H_0)
S* ← min(H_0)
while R*_pre ≠ R* do
    S* ← minRel(Post(S*), R*)
    R*_pre ← R*
    R* ← min(S* ∪ R*)
end while
return R*
```

Let us prove that Algorithm 7 is correct.

Theorem 4.57. *Algorithm 7 returns $min(Reach(\mathcal{H}))$.*

Proof. Let us note S_i^* and R_i^* the respective values of variables S^* and R^* of Algorithm 7 after the ith passing in the *while* loop. We will prove by induction that $\uparrow Post(S_i^*) \cup \uparrow R_i^* = \uparrow Post(R_i^*) \cup \uparrow R_i^*$.

<u>Basis:</u> $\uparrow Post(S_0^*) \cup \uparrow R_0^* = \uparrow Post(\min(H_0)) \cup \uparrow \min(H_0) = \uparrow Post(R_0^*) \cup \uparrow R_0^*$.
<u>Induction:</u> let us suppose that, $\forall 0 \leqslant i \leqslant j, \uparrow Post(S_i^*) \cup \uparrow R_i^* = \uparrow Post(R_i^*) \cup \uparrow R_i^*$.
We will show that $\uparrow Post(S_{j+1}^*) \cup \uparrow R_{j+1}^* = \uparrow Post(R_{j+1}^*) \cup \uparrow R_{j+1}^*$.
Let us first remark that, by definition of $minRel$:

$$\uparrow S_{j+1}^* = \uparrow minRel(Post(S_j^*), R_j^*) = \uparrow (Post(S_j^*) \backslash \uparrow R_j^*) \tag{4.7}$$

We so have:

$$\begin{array}{lll} & \uparrow R_{j+1}^* & \\ = & \uparrow (\min(\uparrow (Post(S_j^*) \backslash \uparrow R_j^*) \cup \uparrow R_j^*)) & \text{(observing Algorithm 7)} \\ = & \uparrow (Post(S_j^*) \backslash \uparrow R_j^*) \cup \uparrow R_j^*) & \text{(by definition of } \uparrow \text{ and } \min\text{)} \\ = & \uparrow (Post(S_j^*)) \cup \uparrow R_j^* & \\ = & \uparrow (Post(R_j^*)) \cup \uparrow R_j^* & \text{(by induction hypothesis).} \end{array}$$

Hence:

$$\begin{array}{ll} & \uparrow Post(S_{j+1}^*) \cup \uparrow R_{j+1}^* \\ = & \uparrow Post\big(\uparrow (Post(S_j^*) \backslash \uparrow R_j^*)\big) \cup \uparrow (Post(R_j^*)) \cup \uparrow R_j^* \\ & \text{(thanks to 4.7 and the previous equality over } \uparrow R_{j+1}^*) \\ = & \uparrow Post\big(\uparrow (Post(S_j^*))\big) \cup \uparrow (Post(R_j^*)) \cup \uparrow R_j^* \\ & \text{(see justification } \star \text{ hereunder)} \\ = & \uparrow Post\big(\uparrow (Post(S_j^*) \cup \uparrow R_j^*)\big) \cup \uparrow (Post(R_j^*)) \cup \uparrow R_j^* \\ & \text{(observing } \star\text{, we only add elements present in } \uparrow (Post(R_j^*)) \text{ yet)} \\ = & \uparrow Post\big(\uparrow (Post(R_j^*) \cup \uparrow R_j^*)\big) \cup \uparrow (Post(R_j^*)) \cup \uparrow R_j^* \\ & \text{(by induction hypothesis)} \\ = & \uparrow Post(R_{j+1}^*) \cup \uparrow R_{j+1}^* \\ & \text{(thanks to the previous equality over } \uparrow R_{j+1}^*). \end{array}$$

Justification $\star$:
it is clear that:

$$\begin{array}{ll} & \uparrow Post\big(\uparrow (Post(S_j^*) \backslash \uparrow R_j^*)\big) \cup \uparrow (Post(R_j^*)) \cup \uparrow R_j^* \\ \subseteq & \uparrow Post\big(\uparrow (Post(S_j^*))\big) \cup \uparrow (Post(R_j^*)) \cup \uparrow R_j^*. \end{array}$$

To show the other inclusion, let us consider $x \in \uparrow Post\big(\uparrow (Post(S_j^*))\big)$ emanating from an element $y \in Post(S_j^*) \cap \uparrow R_j^*$ (the only added elements). We have:

$$\begin{aligned} & \uparrow Post(\uparrow y) \\ = \; & \uparrow Post(y) \qquad \text{(by Lemma 4.50)} \\ \subseteq \; & \uparrow Post(\uparrow R_j^*) \\ = \; & \uparrow Post(R_j^*) \qquad \text{(by Lemma 4.50)}, \end{aligned}$$

and so, $x \in \uparrow Post(R_j^*)$ and hence $x \in \uparrow Post\big(\uparrow (Post(S_j^*) \backslash \uparrow R_j^*)\big) \cup \uparrow (Post(R_j^*)) \cup \uparrow R_j^*$.

Now, let us note $\tilde{R}_i$ the value of variable $\tilde{R}$ of Algorithm 6 after the ith passing in this *while* loop. We show by induction that, $\forall i \geqslant 0$: $R_i^* = \tilde{R}_i$.

<u>Basis:</u> $R_0^* = \min(H_0) = \tilde{R}_0$.

<u>Inductive case:</u> suppose that $\forall 0 \leqslant i \leqslant j$: $R_i^* = \tilde{R}_i$. We will show that $R_{j+1}^* = \tilde{R}_{j+1}$.

$$
\begin{aligned}
R^*_{i+1} &= \min(R^*_i \cup S^*_{i+1}) \\
&\text{(observing Algorithm 7)} \\
&= \min\Big(R^*_i \cup \uparrow (Post(S^*_i) \backslash \uparrow R^*_i)\Big) \\
&\text{(by what we previously saw)} \\
&= \min\Big(\uparrow R^*_i \cup \uparrow (Post(S^*_i) \backslash \uparrow R^*_i)\Big) \\
&= \min\Big(\uparrow R^*_i \cup \uparrow (Post(S^*_i))\Big) \\
&= \min\Big(\uparrow R^*_i \cup \uparrow (Post(R^*_i))\Big) \\
&\text{(by the result of the previous induction)} \\
&= \min\Big(\uparrow \tilde{R}_i \cup \uparrow (Post(\tilde{R}_i))\Big) \\
&\text{(by induction hypothesis)} \\
&= \min\Big(\uparrow \min\big(\uparrow \textstyle\bigcup_{j=0}^{i} Post^j(H_0)\big) \cup \uparrow \Big(Post\big(\min(\uparrow \textstyle\bigcup_{j=0}^{i} Post^j(H_0))\big)\Big)\Big) \\
&\text{(see Algorithms 6 and 5)} \\
&= \min\Big(\uparrow \min\big(\uparrow \textstyle\bigcup_{j=0}^{i} Post^j(H_0)\big) \cup \uparrow \Big(\min\Big(Post\big(\min\big(\uparrow \textstyle\bigcup_{j=0}^{i} Post^j(H_0)\big)\big)\Big)\Big)\Big) \\
&= \min\Big(\uparrow \min\big(\uparrow \textstyle\bigcup_{j=0}^{i} Post^j(H_0)\big) \cup \uparrow \Big(\min\Big(Post\big(\uparrow \textstyle\bigcup_{j=0}^{i} Post^j(H_0)\big)\Big)\Big)\Big) \\
&\text{(by Lemma 4.53)} \\
&= \min\Big(\uparrow \textstyle\bigcup_{j=0}^{i} Post^j(H_0) \cup \uparrow Post\big(\uparrow \textstyle\bigcup_{j=0}^{i} Post^j(H_0)\big)\Big) \\
&= \min\Big(\uparrow \textstyle\bigcup_{j=0}^{i} Post^j(H_0) \cup \uparrow \textstyle\bigcup_{j=1}^{i+1} Post^j(H_0)\Big) \\
&= \min\Big(\uparrow \textstyle\bigcup_{j=0}^{i+1} Post^j(H_0)\Big) \\
&= \tilde{R}_{i+1} \\
&\text{(see Algorithms 5 and 6).}
\end{aligned}
$$

This ends the proof of our theorem. Indeed, Algorithm 7 will return the same output as Algorithm 6, i.e. $\min(Reach(\mathcal{H}))$, in the same number of passing in the *while* loop, thanks to the fact that, $\forall i \geqslant 0$, $R^*_i = \tilde{R}_i$. □

Thanks to this improved algorithm to compute the minimal elements of $Reach(\mathcal{H})$, we can easily produce an MITL model-checking algorithm. We present Algorithm 8 that works *on the fly*, stopping as soon as an accepting word of $\mathcal{H}$ is reached.

Algorithm 8 ImprovedMITLModelChecking

Input: H_0

Output: true iff $\mathcal{B} \models \Phi$.

```
1:  R*_pre ← ∅
2:  R* ← min(H_0)
3:  S* ← min(H_0)
4:  while R*_pre ≠ R* do
5:      S_post ← ∅
6:      for s ∈ S* do
7:          if s is accepting then
8:              return false
9:          else
10:             S_post ← S_post ∪ Post(s)
11:         end if
12:     end for
13:     S* ← minRel(S_post, R*)
14:     R*_pre ← R*
15:     R* ← min(S* ∪ R*)
16: end while
17: return R*
```

4.4 Zone-based algorithm

In the case of TAs, zones have been advocated as a data structure which is more efficient in practice than regions [4, 8, 27]. This fact has been experimentally verified, especially by the multitude of tests provided by the tool UPPAAL [41]. In this section, we will show how zones for OCATA [2] can be adapted to represent set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$. Let us recall we fixed an MITL formula Φ, the OCATA $\mathcal{A}_{\neg\Phi} = (\Sigma, L, \ell_0, F, \delta)$ representing the negation of Φ and a TA $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$. Intuitively, a zone is a guard on the values of the clocks and clock copies, with additional information to associate clock copies of $\mathcal{A}_{\neg\Phi}$ and clocks of $\mathcal{B}$ to the locations of these automata they are present in.

We start by defining some notations for the sequel of this section.
We note x the unique clock of $\mathcal{A}_{\neg\Phi}$ and, for $\mathcal{B}$, we note $x_1^{\mathcal{B}}, x_2^{\mathcal{B}}, \ldots, x_n^{\mathcal{B}}$ the clocks of X.
We note *Copies* the set of $M(\neg\Phi)$ copies of x denoted $x_1, x_2, \ldots, x_{M(\neg\Phi)/2}$, $y_1, y_2, \ldots, y_{M(\neg\Phi)/2}$. Intuitively, each pair of clock copies (x_i, y_i) will represent an interval.
We also note:

$$\begin{aligned} Copies_{begin} &:= \{x_1, x_2, \ldots, x_{M(\neg\Phi)/2}\}, \\ Copies_{end} &:= \{y_1, y_2, \ldots, y_{M(\neg\Phi)/2}\}, \end{aligned}$$

for $1 \leqslant m \leqslant M(\neg\Phi)/2$:

$$\begin{aligned} Copies^m &:= \{x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_m\}, \\ Copies^m_{begin} &:= \{x_1, x_2, \ldots, x_m\}, \end{aligned}$$

and $Copies^0(x) = Copies^0_{begin} = \varnothing$.
Our definition of zone uses a supplementary clock x_0 whose value is always 0.

Thanks to these notations, we are now able to formally define a *zone*. Its definition is based on the notion of *extended guard*.

Definition 4.58. *Let X be a set of clocks. Define the set of* extended guards *over X, denoted $\mathcal{G}_{Ext}(X)$, by the following grammar:*

$$\Phi := \top \quad | \quad c_1 - c_2 \bowtie k \quad | \quad \Phi_1 \wedge \Phi_2,$$

where $c_1, c_2 \in X$, $k \in \mathbb{N}$ *and* $\bowtie \in \{<, \leqslant, >, \geqslant\}$.
$c_1 - c_2 \bowtie k$ *is simply called an* extended clock constraint.

In the sequel, we will sometimes shorten $c_1 - c_2 \bowtie 0$ by $c_1 \bowtie c_2$ and $c_1 - x_0 \bowtie k$ by $c_1 \bowtie k$. We will also use $c_1 = c_2$ as shorthand for $(c_1 - c_2 \geqslant 0) \wedge (c_1 - c_2 \leqslant 0)$.

When considering a classical zone, over a timed automaton, it is simply defined as an extended guard over the set of clock of this automaton. The satisfiability or model-checking algorithm then only need to maintain a zone and the location of the timed automaton in which is present the system at each step.
Now, we are considering a timed automaton *and* an OCATA, to perform model-checking. Hence, we must take care of several copies of the clock of our OCATA: those copies may be present in *different* locations of our OCATA *in the same time*. We so need to maintain in a new component of our definition of zone a function giving, for each clock copy of our OCATA, the location it is in. For the sake of continuity in our definition, we also add a component giving the location of the timed automaton we are present in.

Definition 4.59. *A zone* $\mathcal{Z}_m$ *is a tuple* $(loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ *where:*

- $m \in \mathbb{N}_0$,
- $loc_{\mathcal{A}} : Copies^m_{begin} \to L$,
- $loc_{\mathcal{B}} \in B$ *is a location of* B, *and*
- Z *is an extended guard on* $Copies^m \cup X \cup \{x_0\}$ *(it is a 'classical zone' on this set of clocks [27]).*

In fact, a zone $\mathcal{Z}_m$ is a symbolic representation of a particular set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$, as defined below.

Definition 4.60. *Let $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ be a zone on set of clocks $Copies^m \cup X = \{x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_m\} \cup \{x_1^{\mathcal{B}}, x_2^{\mathcal{B}}, \ldots, x_n^{\mathcal{B}}\}$, such that $loc_{\mathcal{B}} = loc$.*
Then, we let $[\![\mathcal{Z}_m]\!]$ be the denotation of $\mathcal{Z}_m$ defined as the following set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$:

$$\{ \quad (loc, v(x_1^{\mathcal{B}}), v(x_2^{\mathcal{B}}), \ldots, v(x_n^{\mathcal{B}})),\ (loc^{x_1}, [v(x_1), v(y_1)]),\ \ldots, \\ (loc^{x_m}, [v(x_m), v(y_m)])\ \ \},$$

such that v is a valuation of $Copies^m \cup X$ with $v \models \mathcal{Z}_m$ and $loc^c = loc_{\mathcal{A}}(c)$ for all $c \in Copies^m_{begin}$.
For a set ζ of zones, we note $[\![\zeta]\!] := \bigcup_{\mathcal{Z}_m \in \zeta} [\![\mathcal{Z}_m]\!]$.

By abuse of notation, we sometimes write $s \in \mathcal{Z}_m$ instead of $s \in [\![\mathcal{Z}_m]\!]$.

Remark 4.61. We notice that, in the definition of zone $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$, there is no assumption about the constants to use in the extended guards of Z. In particular, they may contain constants greater than the maximal constants of $\mathcal{A}_{\neg\Phi}$ and $\mathcal{B}$.

Definition 4.62. *The initial zone is $\mathcal{Z}_1^{init} = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ with $Z = x_1^{\mathcal{B}} = 0 \wedge \cdots \wedge x_n^{\mathcal{B}} = 0 \wedge x_1 = 0 \wedge y_1 = 0$, $loc_{\mathcal{A}}(x_1) = \ell_0$ and $loc_{\mathcal{B}} = b_0$.*
A zone $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ is accepting iff $\forall 1 \leqslant i \leqslant m$, $loc_{\mathcal{A}}(x_i) \in F$ and $loc_{\mathcal{B}} \in F^{\mathcal{B}}$.

Example 4.63. Let us consider again the TA $\mathcal{B}$ of Figure 4.12 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 4.11. The initial zone is $\mathcal{Z}_1^{init} = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ with:

- $loc_{\mathcal{A}}(x_1) = \ell_{\Box}$,
- $loc_{\mathcal{B}} = b_0$, and
- $Z = x_1^{\mathcal{B}} = 0 \wedge x_1 = 0 \wedge y_1 = 0$.

The unique state of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ represented by $\mathcal{Z}_1^{init}$ is:

$$\{(b_0, 0), (\ell_{\Box}, [0, 0])\},$$

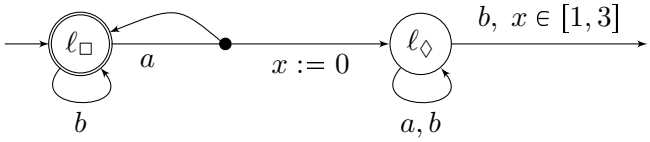


Figure 4.11: OCATA $\mathcal{A}_{\neg\Phi}$ with $\neg\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,3]} b)$.

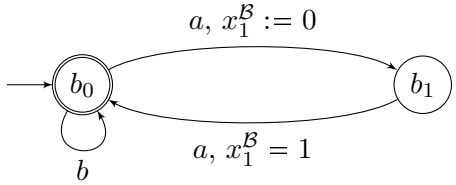


Figure 4.12: A timed automaton $\mathcal{B}$.

it is the initial state of $\mathcal{S}_{\mathcal{B},\neg\Phi}$. $\mathcal{Z}_1^{init}$ is accepting because locations b_0 and $\ell_\Box$ are both accepting.
Here is an example of a zone using 2 pairs of clock copies: $\mathcal{Z}_2 := (loc'_{\mathcal{A}}, loc'_{\mathcal{B}}, Z')$, with:

- $loc'_{\mathcal{A}}(x_1) = \ell_\Box$ and $loc'_{\mathcal{A}}(x_2) = \ell_\Diamond$
- $loc'_{\mathcal{B}} = b_1$, and
- $Z' = x_1 \geqslant 0 \wedge y_1 \geqslant 0 \wedge x_1 = y_1 \wedge x_2 = 0 \wedge y_2 = 0 \wedge x_1^{\mathcal{B}} = 0$.

$\mathcal{Z}_2$ represents the set of states:

$$\{(b_1, 0), (\ell_\Box, [t, t]), (\ell_\Diamond, [0, 0]) \mid t \in \mathbb{R}^+\}.$$

Classical algorithms on zones (over finite words) follows the same steps as that over regions. We start from the initial zone and compute its successors. We look for an *accepting* zone *reachable* from the initial zone. Let us now define the timed and discrete successors of a given zone.

Definition 4.64. *Let $\mathcal{Z}_m$ be a zone. $Post_T(\mathcal{Z}_m)$ denotes the zone such that:*

$$[\![Post_T(\mathcal{Z}_m)]\!] = \{s' \in \mathcal{S}_{\mathcal{B},\neg\Phi} \mid \exists s \in \mathcal{Z}_m \text{ and } t \in \mathbb{R}^+ \text{ such that } s \overset{t}{\rightsquigarrow} s'\}.$$

We can easily compute $Post_T(\mathcal{Z}_m)$ from $\mathcal{Z}_m$. The classical technique consists in, first, putting $\mathcal{Z}_m$ in *normal form*, i.e. to add to Z all the extended clock constraints induced by those it contains (this treatment does not change $[\![\mathcal{Z}_m]\!]$). Then, it remains to delete all the clock constraints of the form $c < k$ or $c \leqslant k$, for all $c \in Copies^m \cup \{x_1^{\mathcal{B}}, x_2^{\mathcal{B}}, \ldots, x_n^{\mathcal{B}}\}$ and all $k \in \mathbb{N}$. (See [11] for details.)

Example 4.65. Let us consider again the TA $\mathcal{B}$ of Figure 4.12, the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 4.11 and the initial zone $\mathcal{Z}_1^{init} = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ (given in Example 4.63) with, once Z is put in normal form:

- $loc_{\mathcal{A}}(x_1) = \ell_{\Box}$,
- $loc_{\mathcal{B}} = b_0$, and
- $Z = x_1^{\mathcal{B}} \leqslant 0 \wedge x_1^{\mathcal{B}} \geqslant 0 \wedge x_1 \leqslant 0 \wedge x_1 \geqslant 0 \wedge y_1 \leqslant 0 \wedge y_1 \geqslant 0 \wedge x_1^{\mathcal{B}} \leqslant x_1 \wedge x_1^{\mathcal{B}} \geqslant x_1 \wedge x_1^{\mathcal{B}} \leqslant y_1 \wedge x_1^{\mathcal{B}} \geqslant y_1 \wedge x_1 \leqslant y_1 \wedge x_1 \geqslant y_1$.

Then, $Post_T(\mathcal{Z}_m)$ is $(loc'_{\mathcal{A}}, loc'_{\mathcal{B}}, Z')$ with:

- $loc_{\mathcal{A}}(x_1) = \ell_{\Box}$,
- $loc_{\mathcal{B}} = b_0$, and
- $Z = x_1^{\mathcal{B}} \geqslant 0 \wedge x_1 \geqslant 0 \wedge y_1 \geqslant 0 \wedge x_1^{\mathcal{B}} \leqslant x_1 \wedge x_1^{\mathcal{B}} \geqslant x_1 \wedge x_1^{\mathcal{B}} \leqslant y_1 \wedge x_1^{\mathcal{B}} \geqslant y_1 \wedge x_1 \leqslant y_1 \wedge x_1 \geqslant y_1$.

Definition 4.66. *Let $\mathcal{Z}_m$ be a zone. $Post_D(\mathcal{Z}_m)$ denotes a set of zones $\mathcal{Z}$ such that:*

$$[\![Post_T(\mathcal{Z}_m)]\!] = \{s' \in \mathcal{S}_{\mathcal{B},\neg\Phi} \mid \exists s \in \mathcal{Z}_m \text{ and } \sigma \in \Sigma \text{ such that } s \xrightarrow{\sigma} s'\}.$$

Remark 4.67. We notice that a discrete successor of a state s of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ does not necessarily use the same number of clock copies as s. Hence, an element of $Post_D(\mathcal{Z}_m)$ may be a zone using less or more than m pairs of clock copies.

We now give two examples of computations of $Post_D(\mathcal{Z}_m)$. Then, we give an intuition on the general way to determine it and finally, we formally define how to compute $Post_D(\mathcal{Z}_m)$.

Example 4.68. Let us consider again the TA $\mathcal{B}$ of Figure 4.12 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 4.11. We consider the zone $\mathcal{Z}_1 := (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ with:

- $loc_{\mathcal{A}}(x_1) = \ell_\Box$,
- $loc_{\mathcal{B}} = b_0$, and
- $Z = x_1^{\mathcal{B}} \geqslant 0 \wedge x_1 \geqslant 0 \wedge y_1 \geqslant 0 \wedge x_1 = y_1 \wedge x_1 = x_1^{\mathcal{B}}$.

We are looking for $Post_D(\mathcal{Z}_1)$. Let $s \in \mathcal{Z}_1$. s is of the form:

$$\{(b_0, t), (\ell_\Box, [t, t])\}, \text{ for some } t \in \mathbb{R}^+.$$

On the one hand, a discrete successor of such an s, reading $b \in \Sigma$, will be a state of the form:

$$\{(b_0, t), (\ell_\Box, [t, t])\}.$$

This set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is exactly represented by the zone $\mathcal{Z}_1$ itself. On the other hand, a discrete successor of such an s, reading $a \in \Sigma$, will be a state of the form:

$$\{(b_1, 0), (\ell_\Box, [t, t]), (\ell_\Diamond, [0, 0])\}, \text{ for } t \in \mathbb{R}^+.$$

This set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is exactly represented by the zone $\mathcal{Z}_2^a := (loc'_{\mathcal{A}}, loc'_{\mathcal{B}}, Z')$, with:

- $loc'_{\mathcal{A}}(x_1) = \ell_\Box$ and $loc'_{\mathcal{A}}(x_2) = \ell_\Diamond$

- $loc'_{\mathcal{B}} = b_1$, and
- $Z' = x_1 \geqslant 0 \wedge y_1 \geqslant 0 \wedge x_1 = y_1 \wedge x_2 = 0 \wedge y_2 = 0 \wedge x_1^{\mathcal{B}} = 0$.

We conclude that $Post_D(\mathcal{Z}_1) := \{\mathcal{Z}_1, \mathcal{Z}_2^a\}$. $Post_D(\mathcal{Z}_1)$ contains one zone using 1 pair of clock copies (i.e. as many as $\mathcal{Z}_1$) and one zone using 2 pairs of clock copies (i.e. more than $\mathcal{Z}_1$).

Example 4.69. Let us still consider the TA $\mathcal{B}$ of Figure 4.12 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 4.11. We consider the zone $\mathcal{Z}_2 := (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ with:

- $loc_{\mathcal{A}}(x_1) = \ell_\Box$ and $loc_{\mathcal{A}}(x_2) = \ell_\Diamond$,
- $loc_{\mathcal{B}} = b_1$, and
- $Z = x_1 \geqslant 1 \wedge y_1 \geqslant 1 \wedge x_1 = y_1 \wedge x_2 = 1 \wedge y_2 = 1 \wedge x_1^{\mathcal{B}} = 1$.

We are looking for $Post_D(\mathcal{Z}_2)$. Let us consider $s \in \mathcal{Z}_2$. s is of the form:

$$\{(b_1, 1), (\ell_\Box, [t, t]), (\ell_\Diamond, [1, 1])\}, \text{ for some } t \geqslant 1.$$

Remark that we can only read an a from such a state (see the unique arc starting from b_1 in $\mathcal{B}$). A discrete successor of s can either be of the form:

(i) $\{(b_0, 1), (\ell_\Box, [t, t]), (\ell_\Diamond, [0, 0]), (\ell_\Diamond, [1, 1])\}$, or

(ii) $\{(b_0, 1), (\ell_\Box, [t, t]), (\ell_\Diamond, [0, 1])\}$.

We note that case (i) is only possible because $M(\neg\Phi)$ is big enough. The set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ presented in (i) is exactly represented by the zone $\mathcal{Z}_3 := (loc^3_{\mathcal{A}}, loc^3_{\mathcal{B}}, Z^3)$ with:

- $loc^3_{\mathcal{A}}(x_1) = \ell_\Box$, $loc^3_{\mathcal{A}}(x_2) = \ell_\Diamond$ and $loc^3_{\mathcal{A}}(x_3) = \ell_\Diamond$,
- $loc^3_{\mathcal{B}} = b_0$, and

- $Z^3 = x_1 \geqslant 1 \wedge y_1 \geqslant 1 \wedge x_1 = y_1 \wedge x_2 = 0 \wedge y_2 = 0 \wedge x_3 = 1 \wedge y_3 = 1 \wedge x_1^{\mathcal{B}} = 1$.

The set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ presented in (ii) is exactly represented by the zone $\mathcal{Z}_2^{group} := (loc_{\mathcal{A}}^{group}, loc_{\mathcal{B}}^{group}, Z^{group})$ with:

- $loc_{\mathcal{A}}^{group}(x_1) = \ell_{\Box}$ and $loc_{\mathcal{A}}^{group}(x_2) = \ell_{\Diamond}$,
- $loc_{\mathcal{B}}^{group} = b_0$, and
- $Z^{group} = x_1 \geqslant 1 \wedge y_1 \geqslant 1 \wedge x_1 = y_1 \wedge x_2 = 0 \wedge y_2 = 1 \wedge x_1^{\mathcal{B}} = 1$.

We conclude that $Post_D(\mathcal{Z}_2) := \{\mathcal{Z}_3, \mathcal{Z}_2^{group}\}$. $Post_D(\mathcal{Z}_2)$ contains one zone using 3 pairs of clock copies ($\mathcal{Z}_3$) and one zone using 2 pairs of clock copies ($\mathcal{Z}_2^{group}$).

In general, we compute $Post_D(\mathcal{Z}_m)$, where $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$, as follows:

- for each label $\sigma \in \Sigma$,
- for each possible transition $t_{\mathcal{B}}$ labelled by σ of $\mathcal{B}$, starting from location $loc_{\mathcal{B}}$,
- for each possible combination of transitions of $\mathcal{A}_{\neg\Phi}$ $(t_1, \ldots, t_m)$, all labelled by σ, such that $\forall 1 \leqslant i \leqslant m$, t_i starts from $loc_{\mathcal{A}}(x_i)$[3],

the possible successor zones are found following the arc $t_{\mathcal{B}}$ from $loc_{\mathcal{B}}$, t_1 from $loc_{\mathcal{A}}(x_1)$, ..., and t_m from $loc_{\mathcal{A}}(x_m)$. First, to take $t_{\mathcal{B}}$ implies to satisfy the clock constraint it carries: these clock constraints must be added to those of Z on $\{x_1^{\mathcal{B}}, x_2^{\mathcal{B}}, \ldots, x_n^{\mathcal{B}}\}$ (it can be easily done on Z using a well-known algorithm on classical zones [11]). Location $loc_{\mathcal{B}}$ must also be modified in the location $t_{\mathcal{B}}$ goes to and, potentially, certain clocks among $\{x_1^{\mathcal{B}}, x_2^{\mathcal{B}}, \ldots, x_n^{\mathcal{B}}\}$ need to be reset (once again, it can be easily done using a well-known algorithm on classical zones [11]).

[3] they associate to each interval (x_i, y_i) a transition t_i to take

Second, to take transitions t_i (for $1 \leqslant i \leqslant m$) implies that (x_i, y_i) must satisfy the clock constraints it carries: these clock constraints must be added to those of $\mathcal{Z}_m$ on *Copies*. Moreover, to take transitions $t_1, \ldots, t_m$ can create new clock copies of value 0 in certain locations or/and letting clock copies with the same zone constraints in the same locations. The new copies with value 0 that have just been created in certain locations can either:

- be grouped with the previous smallest interval associated to this location (see Example 4.69: the states of point (ii) and the associated zone $\mathcal{Z}_2^{group}$)

  $\rightsquigarrow$ let us call ℓ this location: it corresponds to reset the clock x_i such that, in $\mathcal{Z}_m$, $loc_{\mathcal{A}}(x_i) = \ell$ and t_i loops on ℓ for a clock copy x_i with a minimum value.
  (We can only do this if there were such an interval associated to this location.)

- create a new interval [0,0] associated to this location in the zone (see Example 4.68 for the creation of the zone $\mathcal{Z}_2^a$; and see Example 4.69: the states of point (i) and the associated zone $\mathcal{Z}_3$)

  $\rightsquigarrow$ let us call ℓ this location: it corresponds to use two new unused clock copies of x, x_{m+1} and y_{m+1}, and extend function $loc_{\mathcal{A}}$ in way $loc_{\mathcal{A}}(x_{m+1}) = \ell$. Clock copies x_{m+1} and y_{m+1} must be reset.
  (We can only do this if the new number of intervals associated to the locations of $\mathcal{A}_{\neg\Phi}$ does not exceed $M(\neg\Phi)/2$, see Theorem 4.7.)

In the sequel, we formally define what we have just intuitively explained. Let us fix a zone $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$. We will use the following formal notations.

- For an arc t of $\mathcal{A}_{\neg\Phi}$ of the form $(\ell, \sigma, \bigwedge_k A_{j,k})$ such that $\bigwedge_k A_{j,k}$ is a disjunct in $\delta(\ell, \sigma)$, we let $\mathsf{dest}(t) = \{\ell \mid \exists k : A_{j,k} = \ell \vee A_{j,k} = x.\ell\}$ to be the set of destinations of t.

- We first collect the arcs labelled by σ that start in the current locations of both automata and combine them:

  1. $\forall c \in Copies^m$, $\forall \sigma \in \Sigma$:
     $E(c, \sigma) := \{$ arcs labelled by σ and whose starting location is $loc_{\mathcal{A}}(c)\}$,
  2. $\forall \sigma \in \Sigma$:
     $E(\mathcal{B}, \sigma) := \{$arcs starting from $loc_{\mathcal{B}}$ and labelled by $\sigma\}$,
  3. $\forall \sigma \in \Sigma$:
     $Z \odot \sigma := E(\mathcal{B}, \sigma) \times \prod_{i=1}^{m} E(x_i, \sigma)$.

- Then, we introduce definitions that will allow us to select from the zone the valuations that satisfy the guards of selected arcs from $\mathcal{A}_{\neg\Phi}$ and $\mathcal{B}$. To do this, we collect the guards of all these arcs and combine them:

  1. for every arc t:
     $Constr(t) := \{c \mid c$ is a clock constraint in the guard of $t\}$,
  2. for $t_{\mathcal{B}} = (b_{start}, \sigma, g, r, b_{arrival})$ an arc of $\mathcal{B}$, and for a sequence $t_1, \ldots, t_m$ of arcs of $\mathcal{A}_{\neg\Phi}$, we let:

$$g_{t_{\mathcal{B}}, t_1, \ldots, t_m} := g \quad \wedge \bigwedge_{\substack{1 \leqslant i \leqslant m \\ c_i \in Constr(t_i)}} \left(c_i|_{x=x_i} \wedge c_i|_{x=y_i} \right).$$

- Now, we fix an arc $t_{\mathcal{B}} = (b_{start}, \sigma, g, r, b_{arrival})$ of $\mathcal{B}$, and a sequence $t_1, \ldots, t_m$ of arcs of $\mathcal{A}_{\neg\Phi}$ to be fired simultaneously. We introduce definitions that allow us to compute the locations (and the clock copies present in them) that will be active after firing these arcs. We need to distinguish between locations on which we loop from others. Indeed, when we do not loop, we must reset one (see Example 4.69, point (ii) and zone $\mathcal{Z}_3$) or two (see Example 4.69, point (i) and zone $\mathcal{Z}_2^{group}$) clock(s) in the destination location.

  1. We first define the following set keeping the destination location of $t_{\mathcal{B}}$ and of each clock copy of $\mathcal{A}_{\neg\Phi}$ that changes of location:

$$\begin{aligned} LOC(t_{\mathcal{B}}, t_1, \ldots, t_m) \ := \ & \{b_{arrival}\} \cup \\ & \{\, loc \mid \exists 1 \leqslant i \leqslant m : loc \in \mathsf{dest}(t_i) \backslash loc_{\mathcal{A}}(x_i)\}. \end{aligned}$$

2. We define a set containing the clock copies that loop on their location:

$$\begin{aligned} Loop(t_1, \ldots, t_m) \ := \ & \{x_i \mid loc_{\mathcal{A}}(x_i) \in \mathsf{dest}(t_i)\} \\ & \cup \{y_i \mid loc_{\mathcal{A}}(x_i) \in \mathsf{dest}(t_i)\}. \end{aligned}$$

3. From $Loop(t_1, \ldots, t_m)$, we need to extract the clock copies with the minimal value associated to each location ℓ on which some clock copies loop. Indeed, in case of a merging of intervals in ℓ, such a clock copy will be reset (see Example 4.69, point (i) and zone $\mathcal{Z}_2^{group}$):

$$\begin{aligned} & minLoop(t_1, \ldots, t_m) \ := \\ & \quad \{x_i \in Loop(t_1, \ldots, t_m) \cap Copies_{begin} \mid \\ & \quad \forall x_i' \in Loop(t_1, \ldots, t_m) \text{ with } loc_{\mathcal{A}}(x_i) = loc_{\mathcal{A}}(x_i'), \\ & \quad x_i' \geqslant x_i \text{ is implied by the extended guard of } Z_m\}. \end{aligned}$$

- Finally, to conclude the effect of the combined firing of $(t_{\mathcal{B}}, t_1, \ldots, t_m)$, we need to compute which clocks and clock copies will be reset. By the previous definitions, we only need to reset one or two clock copie(s) in the locations $\ell \in LOC(t_{\mathcal{B}}, t_1, \ldots, t_m)$.
  For each such location ℓ, we note r^ℓ an element of the set $\{\{x^\ell\}, \{x, y\}\}$ such that:

  (a) $r^\ell = \{x^\ell\}$ implies that $x^\ell \in minLoop(t_1, \ldots, t_m)$ and $loc_{\mathcal{A}}(x^\ell) = \ell$, and

  (b) $r^\ell = \{x, y\}$ implies that $x \in Copies_{begin} \backslash Loop(t_1, \ldots, t_m)$ and $y \in Copies_{end} \backslash Loop(t_1, \ldots, t_m)$.

  we furthermore require that

  (c) for all $\ell_i, \ell_j \in LOC(t_{\mathcal{B}}, t_1, \ldots, t_m)$: $r^{\ell_i} \cap r^{\ell_j} = \varnothing$.

Thanks to those notations, we are now able to define the elements of $Post_D(\mathcal{Z}_m)$.

For $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$,

$$\mathcal{Z}'_{m'} = (loc'_{\mathcal{A}}, loc'_{\mathcal{B}}, Z') \in Post_D(\mathcal{Z}_m)$$
iff

there exists $\sigma \in \Sigma$, $(t_{\mathcal{B}}, t_1, \ldots, t_m) \in Z \odot \sigma$ and r^ℓ, for all $\ell \in LOC(t_{\mathcal{B}}, t_1, \ldots, t_m)$, such that $g_{t_{\mathcal{B}}, t_1, \ldots, t_m} \cap Z$ is satisfiable, and:

- $Z' = (g_{t_{\mathcal{B}}, t_1, \ldots, t_m} \cap Z) \left[\left(r \cup \bigcup r^\ell\right) := 0\right]$,
- $\forall x_i \in Loop(t_1, \ldots, t_m)$, $loc'_{\mathcal{A}}(x_i)$ is $loc_{\mathcal{A}}(x_i)$,
  for all $\ell \in LOC(t_{\mathcal{B}}, t_1, \ldots, t_m)$, for all $x_\ell \in r^\ell \cap Copies_{begin}$: $loc'_{\mathcal{A}}(x_\ell) = \ell$;
- $loc'_{\mathcal{B}}$ is the destination location of $t_{\mathcal{B}}$.

The following proposition ensures that the elements of $Post_D(\mathcal{Z}_m)$ were correctly defined.

Proposition 4.70. Let $\mathcal{Z}_m$ be a zone.

$$[\![Post_D(\mathcal{Z}_m)]\!] = \{s' \mid \exists s \in \mathcal{Z}_m \text{ such that } s \to s' \text{ in } \mathcal{S}_{\mathcal{B},\neg\Phi}\}.$$

Proof. ($\subseteq$) Suppose that $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$, where $\forall 1 \leqslant k \leqslant m$, $loc_{\mathcal{A}}(x_k) = \ell_k$. Let $s' \in Post_D(\mathcal{Z}_m)$. There exists a certain $s' \in \mathcal{Z}'_{m'}$ for a certain $\mathcal{Z}'_{m'} \in Post_D(\mathcal{Z}_m)$ constructed thanks to $\sigma, t_{\mathcal{B}}, t_1, \ldots, t_m$, and (without loss of generality) thanks to $r^{\ell_1}, \ldots, r^{\ell_p}$, representing respectively the resets in locations $\ell_1, \ldots, \ell_p$. We note $\mathcal{Z}'_{m'} = (loc'_{\mathcal{A}}, loc'_{\mathcal{B}}, Z')$ and $t_{\mathcal{B}} = (b_{start}, \sigma, g, r, b_{arrival})$. Let us suppose that $s' = \{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\} \cup \{(b', v')\}$, with, $\forall 1 \leqslant k' \leqslant m'$, $I'_{k'} = [v'(x_{k'}), v'(y_{k'})]$. We will construct a particular state s of $\mathcal{Z}_m$ and then prove that $s \xrightarrow{\sigma} s'$ in $\mathcal{S}_{\mathcal{B},\neg\Phi}$. Let us construct $s = \{(\ell_k, I_k,)_{k=1}^{m}\} \cup \{(b, v)\}$, with, $\forall 1 \leqslant k \leqslant m$, $I_k = [v(x_k), v(y_k)]$, where:

1. <u>Arc of $\mathcal{B}$ without reset:</u> $\forall 1 \leqslant i \leqslant n$: if $x_i^{\mathcal{B}} \notin r$, we define $v(x_i^{\mathcal{B}}) = v'(x_i^{\mathcal{B}})$,
2. <u>New complete interval:</u> $\forall 1 \leqslant k \leqslant m$: if there exists $1 \leqslant j \leqslant p$ such that r^{ℓ_j} is a doubloon and $x_k \in r^{\ell_j}$, then x_k and y_k were not used in $\mathcal{Z}_m$ and their values must not be defined,

3. Loop without merge: $\forall 1 \leqslant k \leqslant m$: if $\forall 1 \leqslant j \leqslant p$, $x_k \notin r^{\ell_j}$ but that $loc'_{\mathcal{A}}$ is defined on x_k, we define $v(x_k) = v'(x_k)$ and $v(y_k) = v'(y_k)$,

4. Loop with merge: $\forall 1 \leqslant k \leqslant m'$: if there exists $1 \leqslant j \leqslant p$ such that r^{ℓ_j} is a singleton and $x_k \in r^{\ell_j}$, then we define $v(y_k) = v'(y_k)$,

5. Arc going out or arc of $\mathcal{B}$ with reset: the values of $v(x_k)$, $v(y_k)$ and $v(x_i^{\mathcal{B}})$ that we still must define are arbitrarily chosen in way they satisfy the extended clock constraints of $g_{t_{\mathcal{B}},t_1,\ldots,t_m} \cap Z$ (which is possible because $g_{t_{\mathcal{B}},t_1,\ldots,t_m} \cap Z$ is satisfiable and we only chose values of clocks/clock copies that have the same value in $\mathcal{Z}'_{m'}$, so that they cannot prevent $g_{t_{\mathcal{B}},t_1,\ldots,t_m} \cap Z$ from being satisfiable).

We must prove that $s \xrightarrow{\sigma} s'$ in $\mathcal{S}_{\mathcal{B},\neg\Phi}$ (see Definition 4.17), i.e.: $(b,v) \xrightarrow{\sigma} (b',v')$ in $\mathcal{B}$ and $\{(\ell_k, I_k)_{k=1}^m\} \xrightarrow{\sigma}_{f^\star_{\neg\Phi}} \{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ in $\mathcal{A}_{\neg\Phi}$.
We first show that $(b,v) \xrightarrow{\sigma} (b',v')$ in $\mathcal{B}$ thanks to $t_{\mathcal{B}}$. Indeed, $v \models g$ because g is contained in $g_{t_{\mathcal{B}},t_1,\ldots,t_m} \cap Z$; $\forall x \in r$, $v'(x) = 0$ because it is reset by definition of $\mathcal{Z}'_{m'} \in Post_D(\mathcal{Z}_m)$ and $\forall x \in \{x_1^{\mathcal{B}}, \ldots, x_2^{\mathcal{B}}\} \backslash r$, $v'(x) = v(x)$ by previous point 1..
Now, let us show that $\{(\ell_k, I_k)_{k=1}^m\} \xrightarrow{\sigma}_{f^\star_{\neg\Phi}} \{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ in $\mathcal{A}_{\neg\Phi}$. We must prove there exists minimal models M_k, for $1 \leqslant k \leqslant m$ of $\delta(\ell_k, \sigma)$ with respect to I_k such that $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\} \in f^\star_{\neg\Phi}\left((\bigcup_{k=1}^m M_k)^{\neq}\right)$. For each $1 \leqslant k \leqslant m$, we take M_k to be the minimal model of $\delta(\ell_k, \sigma)$ with respect to I_k obtained following the arc t_k: it can be taken because its clock constraint is verified on $v(x_k)$ and $v(y_k)$ (it is indeed present in $g_{t_{\mathcal{B}},t_1,\ldots,t_m} \cap Z$) and as this clock constraint can only be an interval (convex), it is also verified on each $i \in I_k$. We then take the element E of $f^\star_{\neg\Phi}\left((\bigcup_{k=1}^m M_k)^{\neq}\right)$ that merges the two more little intervals present in ℓ_j (for $1 \leqslant j \leqslant p$) iff r^{ℓ_j} is a singleton. It remains to prove that the obtained configuration is exactly $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$. It is based on the following facts:

a. each (ℓ_k, I_k) that does not loop disappear from $\{(\ell_k, I_k)_{k=1}^m\}$ to E and is not present in $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$: the clock copies representing such intervals disappear from $\mathcal{Z}_m$ to $\mathcal{Z}'_{m'}$ (i.e. $loc_{\mathcal{A}}$ is not defined on them anymore),

b. for each location ℓ_j with $1 \leqslant j \leqslant p$, destination of at least one t_k for $1 \leqslant k \leqslant m$, $(\ell_j, [0,0])$ is present in $(\bigcup_{k=1}^{m} M_k)^{\neq}$ and

   – <u>if r^{ℓ_j} is a doubloon:</u> two new clock copies, say x_{ℓ_j} and y_{ℓ_j} are used by $\mathcal{Z}'_{m'}$. They are defined and reset in way $(\ell_j, [0,0])$ is also present in $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$;

   – <u>else:</u> by definition of r^{ℓ_j} and $minLoop(t_1, \ldots, t_m)$, the clock copy x_j representing the beginning of the more little interval in $\{(\ell_k, I_k)_{k=1}^{m}\}$ that loops on ℓ_j, say $I_j = [v(x_j), v(y_j)]$, is reset in $\mathcal{Z}'_{m'}$: $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ contains $(\ell_j, [0, v(y_j)])$;

c. each (ℓ_k, I_k) that loops is still present in $(\bigcup_{k=1}^{m} M_k)^{\neq}$: the clock copies representing I_k in $\mathcal{Z}_m$ are still present in $\mathcal{Z}'_{m'}$ but the clock copy representing its beginning could have been reset, so that $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ contains either $(\ell_k, [v(x_k), v(y_k)])$ or $(\ell_k, [0, v(y_k)])$,

d. when computing E from $(\bigcup_{k=1}^{m} M_k)^{\neq}$, we know the two more little intervals present in ℓ_j, for $1 \leqslant j \leqslant p$, are merged iff r^{ℓ_j} is a singleton. So, $(\ell_j, [0,0])$ and (ℓ_j, I_j) are merged into $(\ell_j, [0, v(y_j)])$ iff I_j is the more little interval that loops on ℓ_j, r^{ℓ_j} is a singleton and so must contain the clock copy representing its beginning: x_j. In this case and only in this case, x_j is then reset by definition of $\mathcal{Z}'_{m'} \in Post_D(\mathcal{Z}_m)$ so that E contains $(\ell_j, [0, v(y_j)])$ iff $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ also contains $(\ell_j, [0, v(y_j)])$.

($\supseteq$) Let $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ be a zone and s' be such that $\exists s \in \mathcal{Z}_m$ with $s \to s'$ in $\mathcal{S}_{\mathcal{B},\neg\Phi}$. Let us show that $s' \in [\![Post_D(\mathcal{Z}_m)]\!]$. Let us suppose that $s = \{(\ell_k, I_k)_{k=1}^{m}\} \cup \{(b, v)\}$, with, $\forall 1 \leqslant k \leqslant m$, $I_k = [v(x_k), v(y_k)]$ and $s' = \{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\} \cup \{(b', v')\}$, with, $\forall 1 \leqslant k' \leqslant m'$, $I'_{k'} = [v'(x_{k'}), v'(y_{k'})]$. As $s \to s'$, there exists $\sigma \in \Sigma$ such that:

- $(b, v) \xrightarrow{\sigma} (b', v')$ in $\mathcal{B}$, i.e.: there exists an arc $t_{\mathcal{B}} = (b, \sigma, g, r, b')$ such that $v \models g$, $\forall x \in r$, $v'(x) = 0$ and $\forall x \in \{x_1^{\mathcal{B}}, \ldots, x_n^{\mathcal{B}}\} \backslash r$, $v'(x) = v(x)$;

- $\{(\ell_k, I_k)_{k=1}^m\} \xrightarrow{\sigma}_{f^\star_{\neg\Phi}} \{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ in $\mathcal{A}_{\neg\Phi}$, i.e.: $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\} = E \in f^\star_{\neg\Phi}\left((\bigcup_{k=1}^m M_k)^{\neq}\right)$ for certain minimal models M_k of $\delta(\ell_k, \sigma)$ with respect to I_k, which are themselves obtained by taking certain arcs t_k from ℓ_k, for $1 \leqslant k \leqslant m$.

Let us define r^{ℓ_j}, for $1 \leqslant j \leqslant p$ by:

- r^{ℓ_j} contains two new clock copies *iff* $\exists 1 \leqslant k \leqslant m$ such that t_k goes to ℓ_j with a reset and no merge is applied by $f^\star_{\neg\Phi}$ on ℓ_j,
- r^{ℓ_j} contains the clock representing the beginning of the more little interval present in ℓ_j that loops on ℓ_j taking one of the t_k, for $1 \leqslant k \leqslant m$, *iff* $\exists 1 \leqslant k \leqslant m$ such that t_k goes to ℓ_j with a reset and a merge is applied by $f^\star_{\neg\Phi}$ on ℓ_j,
- r^{ℓ_j} is undefined in the other case, i.e. when none of the t_k, for $1 \leqslant k \leqslant m$, goes to ℓ_j with a reset.

It is easy to prove that the $\mathcal{Z}'_{m'} \in Post_D(\mathcal{Z}_m)$ induced by $\sigma, t_{\mathcal{B}}, t_1, \ldots, t_m, r^{\ell_1}, \ldots, r^{\ell_p}$ contains s' thanks to the following facts:

- as $s \in \mathcal{Z}_m$, the extended clock constraint Z is satisfied by its clock values ; moreover, having taken arcs $t_{\mathcal{B}}, t_1, \ldots, t_m$ ensure the bounds of the intervals and the clock values of s satisfy $g_{t_{\mathcal{B}}, t_1, \ldots, t_m} \cap Z$ (in particular $g_{t_{\mathcal{B}}, t_1, \ldots, t_m} \cap Z$ is satisfiable and $\mathcal{Z}'_{m'}$ really exists),
- a clock $x_i^{\mathcal{B}}$, for $1 \leqslant i \leqslant n$, is reset in the construction of $\mathcal{Z}'_{m'}$ iff $v'(x_i^{\mathcal{B}}) = 0$,
- the facts b., c. and d. of the proof of inclusion $\subseteq$ are still true here.

$\square$

As for the region-based algorithm, we define the operators $Post$, $Post^*$ and $Post^+$ over sets of zones.

Definition 4.71. *Let ζ be a set of zones, we define:*

$$\begin{aligned} Post(\zeta) \quad := \quad \{\mathcal{Z}'_{m'} \mid \; & \exists \sigma \in \Sigma, \text{ a zone } \mathcal{Z}_m \in \zeta \text{ and a zone } \mathcal{Z}''_{m''} \text{ such that:} \\ & \mathcal{Z}''_{m''} \in Post_T(\mathcal{Z}_m) \text{ and } \mathcal{Z}'_{m'} \in Post_D(\mathcal{Z}''_{m''})\}. \end{aligned}$$

Definition 4.72. *Let ζ be a set of zones and $n \in \mathbb{N}_0$, we define: $Post^n(\zeta) = Post(Post^{n-1}(\zeta))$, with $Post^0(\zeta) = \zeta$ and $Post^1(\zeta) = Post(\zeta)$.*
We define $Post^*(\zeta) = \bigcup_{n \in \mathbb{N}} Post^n(\zeta)$ and $Post^+(\zeta) = \bigcup_{n \in \mathbb{N}_0} Post^n(\zeta)$.

Let us recall what was the situation over regions of $\mathcal{H}$. As $\mathcal{H}$ is finite, when considering $\mathcal{W} \subseteq \mathcal{H}$, we had the existence of an $m \in \mathbb{N}$ such that: $Post^*(\mathcal{W}) = \bigcup_{n=1}^{m} Post^n(\mathcal{W})$ (see Definition 5.34).

Unfortunately, as there is no maximal value bounding the constants present in the extended guards of a zone (see Remark 4.61), the number of zones is infinite and hence, in general, so is $Post^*(\zeta)$, for a set of zones ζ. In the litterature, several widening operators have been proposed in way to obtain a finite number of zones to use in algorithms. The correctness of the use of those operators has been subject to debate in the early 2000's. This debate was closed by Patricia Bouyer. In her paper [15], she considers several approximations: not all of them enable to obtain a correct algorithm. In the sequel, we present an MITL model-checking algorithm based on the approximation denoted $Approx_\beta$ presented in [15] and on which reachability algorithms are proved to be correct. When a zone $\mathcal{Z}$ using constants greater than $c_{\max}$ is reached, it is approximated by the smallest zone containing $\mathcal{Z}$ and no extended guard with a constant greater than $c_{\max}$, denoted $Approx_\beta(\mathcal{Z})$ (see [15] for further details). The number of such 'approximated' zones is then finite and an algorithm in the spirit of Algorithm 3 (over regions), will thus terminate.

We can now present Algorithm 9, that uses approximation $Approx_\beta$ and solves the MITL model-checking problem using the same outline as Algorithm 3. This algorithm terminates because there is only a finite number of 'approximated' zones to explore. Its correctness relies on Theorem 2 of [15], proving that using

'approximated zones', according to the operator $Approx_\beta$, is sufficient to obtain a correct reachability algorithm. The proof of this theorem must be slightly adapted to cope with the components $loc_\mathcal{A}$ and $loc_\mathcal{B}$ present in our zones. However, the key elements of the proof only concern the extended guard of zones: from this point of view, the proof of [15] stays correct without any adaptation.

Algorithm 9 MITLModelCheckingWithZones

Input: A TA $\mathcal{B}$ and the ATA $\mathcal{A}_{\neg\Phi}$, for $\Phi \in$ MITL.

Output: 'true' iff $\mathcal{B} \models \Phi$.

```
ToExplore ← Approx_β(Z_1^init)
Explored ← ∅
while ToExplore ≠ ∅ do
    Remove some element Z from ToExplore
    if Z is accepting then
        return 'false'
    end if
    Explored = Explored ∪ {Z}
    ToExplore = ToExplore ∪ (Approx_β(Post(Z)) \ Explored)
end while
return 'true'
```

4.5 Order-based heuristic for zones

As well as for the region setting, we can improve Algorithm 9 thanks to the use of antichains. A simple *inclusion order* is usually used on classical zones (for timed automata). Let us recall that a zone is a compact representation of a set of states of an automaton. When considering two classical zones $\mathcal{Z}$ and $\mathcal{Z}'$, $\mathcal{Z}$ is said *included* in $\mathcal{Z}'$ if the set of states it represents is included in that of $\mathcal{Z}'$. In a reachability algorithm (as Algorithm 9), when we are considering a classical zone $\mathcal{Z}$, while a $\mathcal{Z}'$ in which $\mathcal{Z}$ is included has already been reached, it is not necessary to compute the successors of $\mathcal{Z}$. Indeed, the states they represent are

included in those of the successors of $\mathcal{Z}'$ (already computed).
Let us fix again an MITL formula Φ, the OCATA $\mathcal{A}_{\neg\Phi} = (\Sigma, L, \ell_0, F, \delta)$ representing the negation of Φ and a TA $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$. We consider the timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi}$ of Definition 4.17 and recall that our zones represent a set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$.
The *inclusion order* on classical zones, introduced hereinabove, may be applied to our definition of zone. While classical zones maintain information for a fixed number of clocks, our zones may maintain information from $|X|$ to $|X| + M(\neg\Phi)$ clocks. However, we can consider our zones as classical zones over $|X| + M(\neg\Phi)$ clocks, even if, in each of our zones, several clock copies are unused, *inactive*. We can so easily adapt this inclusion order to our zones.

However, this inclusion order may be improved for our zones, using precisely their particularity of having a variable size. The principle is the same as for the order $\sqsubseteq$ defined on regions (see Definition 4.38). A zone $\mathcal{Z}$ is considered smaller than a zone $\mathcal{Z}'$ if each clock copy (among the $M(\neg\Phi)$ dedicated to the clock copies of $\mathcal{A}_{\neg\Phi}$) of $\mathcal{Z}$ can be *simulated* by a clock copy of $\mathcal{Z}'$. In particular, a clock copy of $\mathcal{Z}$ can only be simulated by a clock copy of $\mathcal{Z}'$ present in the same location of $\mathcal{A}_{\neg\Phi}$. Moreover, if x_1 and x_2 are clock copies of $\mathcal{Z}$, associated to a same location of $\mathcal{A}_{\neg\Phi}$, such that the possible values of x_1 are smaller than that of x_2, then, they can only be (respectively) simulated by clock copies x'_1 and x'_2 of $\mathcal{Z}'$ satisfying the same property.
When such a *simulation* exists between $\mathcal{Z}$ and $\mathcal{Z}'$, the 'inclusion order' introduced hereinabove may be applied to the zone $\mathcal{Z}$ and the subzone of $\mathcal{Z}'$ only consisting in the clock copies that simulate a clock copy of $\mathcal{Z}$. In conclusion, we can define an order $\sqsubseteq_{zones}$ such that $\mathcal{Z} \sqsubseteq_{zones} \mathcal{Z}'$ if

- each clock copy of $\mathcal{Z}$ can be *simulated* by a clock copy of $\mathcal{Z}'$ and
- the subzone of $\mathcal{Z}'$, only consisting in the clock copies that simulate a clock copy of $\mathcal{Z}$, is *included* in $\mathcal{Z}$ (for the inclusion order of classical zones).

A heuristic of algorithm 9 consists in using this order on our zones and maintain-

ing the antichain of the minimal elements obtained this way.

Example 4.73. Let us consider a timed automaton $\mathcal{B}$ without any clock (for the sake of simplicity), and an OCATA $\mathcal{A}$ whose unique clock is x. We moreover consider the three following zones:
1) the zone $\mathcal{Z}_2 = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$, where:

- $loc_{\mathcal{A}}(x_1) = loc_{\mathcal{A}}(x_2) = \ell$,
- $\mathsf{loc}_{\mathcal{B}} = \ell_{\mathcal{B}}$,
- $Z = x_1 = 1 \wedge y_1 = 1 \wedge x_2 = 2 \wedge y_2 = 3$;

2) the zone $\mathcal{Z}_2' = (loc'_{\mathcal{A}}, loc'_{\mathcal{B}}, Z')$, where:

- $loc'_{\mathcal{A}}(x_1) = loc'_{\mathcal{A}}(x_2) = \ell$,
- $\mathsf{loc}'_{\mathcal{B}} = \ell_{\mathcal{B}}$,
- $Z' = x_2 = 1 \wedge y_2 = 1 \wedge x_1 = 2 \wedge y_1 = 3$;

3) the zone $\mathcal{Z}_1 = (loc"_{\mathcal{A}}, loc"_{\mathcal{B}}, Z")$, where:

- $loc"_{\mathcal{A}}(x_1) = \ell$,
- $\mathsf{loc}"_{\mathcal{B}} = \ell_{\mathcal{B}}$,
- $Z" = x_1 = 2 \wedge y_1 = 4$.

On the one hand, we have that $\mathcal{Z}_2 \sqsubseteq_{zones} \mathcal{Z}_2'$. Indeed, the clock copies x_1, y_1, x_2 and y_2 of $\mathcal{Z}_2$ can be respectively simulated by the clock copies x_2, y_2, x_1 and y_1 of $\mathcal{Z}_2'$. In a similar way, $\mathcal{Z}_2' \sqsubseteq_{zones} \mathcal{Z}_2$.[4]

[4]Remark that these clock copies may be swapped because in our setting they are all *copies* of a *unique* clock. When considering classical zones on timed automata, a clock cannot be swapped with another (and so *different*) clock.

On the second hand, we have that $\mathcal{Z}_1 \sqsubseteq_{zones} \mathcal{Z}_2$. Indeed, the clock copies x_1 and y_1 of $\mathcal{Z}_1$ can be respectively simulated by the clock copies x_2 and y_2 of $\mathcal{Z}_2$; then, $\mathcal{Z}_2$ restricted to clock copies x_2 and y_2 is *included* (for the inclusion order of classical zones) in $\mathcal{Z}_1$. Intuitively, when considering a reachability algorithm, if we have already computed the successors of $\mathcal{Z}_1$ and we reach $\mathcal{Z}_2$, then, it is not neccessary to compute the successors of $\mathcal{Z}_2$. Indeed, if we had an accepting run from $\mathcal{Z}_2$, we necessarly also have an accepting run from $\mathcal{Z}_1$ (this is due to the fact that we have an accepting run in an OCATA if *all* its branches are accepting, while we have an accepting run from a given zone $\mathcal{Z}$ if *there exists* a state in $[\![\mathcal{Z}]\!]$ from which we have an accepting run).

CHAPTER 5

MITL satisfiability and model-checking over infinite words

This chapter is dedicated to the setting of *infinite* words. Its aim is similar to that of the previous one: we are looking for algorithms to solve the MITL satisfiability and model-checking problems *over infinite words*. The contributions introduced in this chapter were subject to a publication: [21] (arXiv reference: [22]).

As in the finite words setting, our first goal is to build, from any MITL formula Φ, a *Büchi timed automaton* that accepts $[\![\Phi]\!]^\omega$. Our aim is to obtain this Büchi timed automaton from an OCATA $\mathcal{A}_\Phi$, accepting $[\![\Phi]\!]^\omega$. In fact, we start by showing that, for every MTL formula Φ, the OCATA $\mathcal{A}_\Phi$ obtained by the Ouaknine and Worrell construction ([51], see subsection 2.9) recognises $[\![\Phi]\!]^\omega$ *over infinite words* (a property that had never been established in the case of infinite words, as far as we know[1]). To show that $L(\mathcal{A}_\Phi) = [\![\Phi]\!]$ (over finite words, see

[1] Even in [51] where the authors consider a fragment of MTL over infinite words, but consider only safety properties that are reduced to questions on finite words.

[51]), Ouaknine and Worrell used the key fact that the automaton $\mathcal{A}_\Phi$ is *easily* complementable (in a sense that will be precised later). We obtain this result *over infinite words* showing a similar property of the OCATA $\mathcal{A}_\Phi$. In fact, the automata $\mathcal{A}_\Phi$ obtained from MITL formulas are included in a class of OCATA (we call it the *class of TOCATA*) that are *easily* complementable (in the same sense as in [51]).

From this OCATA $\mathcal{A}_\Phi$ recognising $[\![\Phi]\!]^\omega$, we would like to obtain a Büchi timed automaton $\mathcal{B}_\Phi$, in a similar way as over finite words. However, the construction we gave over finite words must be adapted for the *infinite words* setting. Indeed, a method close to that of Miyano and Hayashi [46], adapted to cope with *timed* words, is required to translate an alternating automaton into a Büchi one (see Definition 2.46 and comments).

Following the same steps as in the finite words case, we finally present our technique to perform 'on the fly' MITL model-checking over infinite words, using directly $\mathcal{A}_\Phi$ to avoid the construction of the whole Büchi timed automaton accepting $[\![\Phi]\!]^\omega$. We present a region-based algorithm as well as a zone-based one. Most of the results stated in this chapter have a twin proposition over finite words in the previous one. The unique difficulty to overtake all along this chapter is the use of markers *à la Miyano Hayashi*, due to the necessary translation from an alternating (timed) automaton to a Büchi (timed) automaton.

5.1 TOCATA: a class of OCATA for MITL

In this section, we are considering infinite words. We start by introducing a strict subclass of OCATA that we call the class of *tree-like* OCATA (TOCATA for short). We are very interested in TOCATA because, as we will show, when translating an MTL formula Φ into the OCATA $\mathcal{A}_\Phi$ (see Section 2.2), one indeed obtains a TOCATA.

Moreover, TOCATA enjoy the useful property to be easily complementable. Indeed, the complement of a TOCATA is another TOCATA, simply obtained switching accepting and non-accepting locations and dualising (see Definition 2.105) the transition relation.

Thanks to this key property, we can adapt the proof of [51] showing that $L(\mathcal{A}_\Phi) = [\![\Phi]\!]$ (over finite words) to obtain the same property over infinite words: $L^\omega(\mathcal{A}_\Phi) = [\![\Phi]\!]^\omega$.

5.1.1 Tree-like OCATA

We define a strict subclass of OCATA that captures all the infinite words languages of MTL formulas. We call it the class of *tree-like* OCATA (TOCATA for short): an appropriate name for OCATA looking like trees.

Definition 5.1. *An OCATA $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ is a TOCATA iff there exists a partition $L_1, L_2, \ldots, L_m$ of L satisfying:*

- *each L_i contains either only accepting locations or no accepting location: $\forall 1 \leqslant i \leqslant m$ either $L_i \subseteq F$ or $L_i \cap F = \emptyset$, and*
- *there is a partial order $\preccurlyeq$ on the sets $L_1, L_2, \ldots, L_m$ compatible with the transition relation and that yields the 'tree-like' structure of the automaton in the following sense: $\preccurlyeq$ is such that $L_j \preccurlyeq L_i$ iff $\exists \sigma \in \Sigma$, $\ell \in L_i$ and $\ell' \in L_j$ such that ℓ' is present in $\delta(\ell, \sigma)$.*

Example 5.2. As an example, let us observe the OCATA of Figure 5.1, with $F = \{\ell_5, \ell_6\}$. It is a TOCATA. In the figure, we organized the locations in way to bring out the tree aspect of this automaton. The locations of this OCATA are partition in L_1, L_2, L_3, L_4, L_5 (dashed rectangles): this partition satisfies the property of Definition 5.1, proving that this automaton is a TOCATA. Indeed, we have:

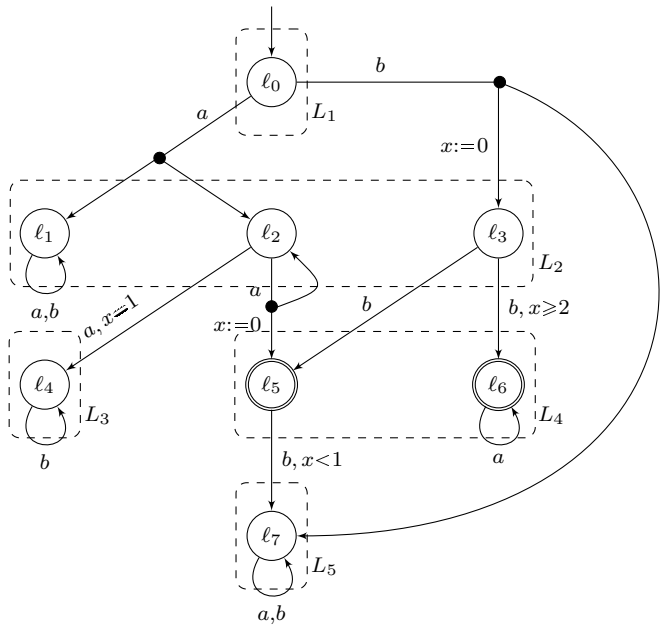


Figure 5.1: A TOCATA example

- $L_1 \cap F = \varnothing$, $L_2 \cap F = \varnothing$, $L_3 \cap F = \varnothing$, $L_4 \subseteq F$ and $L_5 \cap F = \varnothing$;
- $L_3 \preccurlyeq L_2 \preccurlyeq L_1$, $L_5 \preccurlyeq L_4 \preccurlyeq L_2$ and $L_5 \preccurlyeq L_1$.

In particular, OCATA built from *MTL* formulas, are TOCATA. Since MTL is a superset of MITL, this proposition is still true for MITL formulas (in which we are more particularly interested):

Proposition 5.3. For every *MTL* formula Φ, $\mathcal{A}_\Phi$ is a TOCATA.

Proof. Let $L = \{\ell_1, \ell_2, \ldots, \ell_m\}$ be the locations of $\mathcal{A}_\Phi$. We consider the partition $\{\ell_1\}, \{\ell_2\}, \ldots, \{\ell_m\}$ of L and the order $\preccurlyeq$ such that $\{\ell_j\} \preccurlyeq \{\ell_i\}$ iff ℓ_j is a subformula of ℓ_i. It is easy to check that they satisfy the definition of TOCATA. $\square$

Example 5.4. Let us observe the OCATA $\mathcal{A}$ of Figure 5.2, corresponding to the MITL formula $\Phi \equiv \Box\left(a \Rightarrow \left(\Diamond_{[0,1]} b \wedge \Box_{[0,4[}\left(\Diamond_{[0,1]} c\right)\right)\right)$. It is an TOCATA.

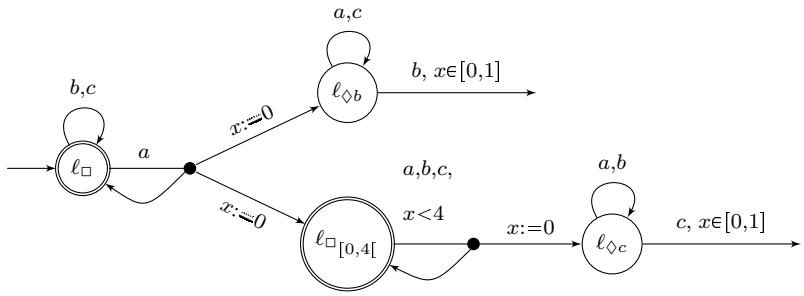


Figure 5.2: An OCATA $\mathcal{A}$ for formula $\Phi \equiv \Box\left(a \Rightarrow \left(\Diamond_{[0,1]} b \wedge \Box_{[0,4[}\left(\Diamond_{[0,1]} c\right)\right)\right)$.

Indeed, using the partition $\{\ell_\Box\}$, $\{\ell_{\Diamond b}\}$, $\{\ell_{\Box_{[0,4[}}\}$ and $\{\ell_{\Diamond c}\}$ on its set of locations, we have:

- $\{\ell_\Box\} \cap F = \varnothing$, $\{\ell_{\Diamond b}\} \cap F = \varnothing$, $\{\ell_{\Box_{[0,4[}}\} \subseteq F$ and $\{\ell_{\Diamond c}\} \cap F = \varnothing$;
- $\Diamond_{[0,1]} b$, $\Box_{[0,4[}\left(\Diamond_{[0,1]} c\right)$ and $\Diamond_{[0,1]} c$ are subformulas of Φ, and their corresponding locations are so 'smaller than' the location of Φ (i.e. $\ell_\Box$): $\{\ell_{\Diamond b}\} \preccurlyeq \{\ell_\Box\}$, $\{\ell_{\Box_{[0,4[}}\} \preccurlyeq \{\ell_\Box\}$ and $\{\ell_{\Diamond c}\} \preccurlyeq \{\ell_\Box\}$. In the same way, $\Diamond_{[0,1]} c$ is a subformula of $\Phi' \equiv \Box_{[0,4[}\left(\Diamond_{[0,1]} c\right)$, and its location is so 'smaller than' that of Φ': $\{\ell_{\Diamond c}\} \preccurlyeq \{\ell_{\Box_{[0,4[}}\}$.

Our definition of TOCATA is close to several classes of automata studied in previous works. In the sequel, we recall these classes of automata and their main characteristics. We start by displaying three classes of *untimed* automata close to our TOCATA. We then focus on one class of *timed* automata comparable to our TOCATA.

Let us begin with the untimed setting. The class of untimed automata studied by Orna Kupferman and Moshe Y. Vardi in [40] (and first defined in [47]) is the class of automata whose definition is the closest to our TOCATA. In [40], the authors are looking at untimed automata they call *weak alternating automata*. These are defined as our TOCATA, but they are untimed alternating automata. The aim of their paper was to present a simple quadratic translation of Büchi

and co-Büchi alternating automata into weak alternating automata.
Our definition of TOCATA can also be compared to a class of automata used by Paul Gastin and Denis Oddoux in [34]. In this paper, they are looking at untimed automata they call *very weak alternating automata*. Such automata are equipped with a partial order over their set of locations L such that $\forall \ell \in L$, all the states appearing in $\delta(\ell, \sigma)$, for a certain $\sigma \in \Sigma$, are lower or equal to ℓ. They use this class of alternating automata to represent LTL formulas: for each LTL formula Ψ, there exists a *very weak alternating automaton* $\mathcal{A}_\Psi$ such that $L^\omega(\mathcal{A}_\Psi) = [\![\Psi]\!]^\omega$. They use this representation of LTL formulas in way to elaborate a translation, from an LTL formula Ψ to a Büchi automaton $\mathcal{B}_\Psi$ recognizing $[\![\Psi]\!]^\omega$, which is efficient in practice. Our procedure to find a timed Büchi automaton $\mathcal{B}_\Phi$ recognizing $[\![\Phi]\!]^\omega$, for a given MITL formula Φ, can be seen as an extention of their work to the timed setting.
In [43], Christof Löding and Wolfgang Thomas observe a class of untimed automata that can also be compared to our TOCATA. In this paper, they are looking at untimed alternating automata with weak parity acceptance condition, that they call *weak parity automata*. The first result of their paper consists in showing the class of *weak parity automata* and the class of *weak alternating automata* of [47] (and [40]) have the same expressiveness. Then, they prove that *weak parity automata* can easily be complemented dualising the transition relation (in an untimed way similar to our Definition 2.105) and adding 1 to the color of each location, in way to also 'dualise' the acceptance condition. A parallel may be done between this result and our method to complement TOCATA presented in the following subsection.

Let us now focus on the timed setting. In [52], Parys and Walukiewicz uses the terms of *weak alternating timed automaton* to designate OCATA with a *weak parity acceptance condition*: they are a timed twins of the *weak parity automata* of [43]. On the one hand, the first result of [43] can be easily adapted to the timed setting in way to prove that the class of *weak alternating timed automata* of [52] and our class of TOCATA have the same expressiveness. On the second hand, a parallel might be done between our work and the results presented in

[52]. Indeed, the authors prove that the emptiness problem is decidable, with a non-primitive recursive complexity, for their *weak alternating timed automata* (see Theorem 2.112). For our part, in the sequel, we will elaborate an algorithm working in 2EXPTIME to solve the MITL satisfiability problem: in fact, this algorithm consists in checking the emptiness of TOCATA representing MITL formulas. Hence, the complexity of the emptiness problem is reduced from the entire class of weak alternating timed automata (or of TOCATA) to that of TOCATA representing MITL formulas. In the same spirit, in [52], the authors made a parallel between their results over weak alternating timed automata and MITL. Indeed, we know that the satisfiability problem is decidable for MITL but undecidable for MTL, so that one can think that to use or not equality in the guards of automata might change decidability results for their emptiness problem. In [52], the authors contradict this intuition showing that it is undecidable if a given one-clock universal[2] timed automaton $\mathcal{A}$ with weak condition $\Omega : L \to \{1, 2\}$ accepts some non Zeno word, even when $\mathcal{A}$ does not use tests for equality. In [52], they argue *'it is not only the lack of punctual constraints, but also the very weak syntax of the logic that makes MITL decidable'*.

5.1.2 Properties of TOCATA

TOCATA enjoy the peculiar property to be easily complementable. One can simply swap accepting and non-accepting locations, and 'dualise' (see Definition 2.105) the transition relation, *without changing the acceptance condition* (as in the case of OCATA over finite words, see [51]). This property will enable us to prove that, being given an MTL formula Φ, the OCATA $\mathcal{A}_\Phi$ of Definition 2.164 (recognizing $[\![\Phi]\!]$, over *finite* words) is such that: $L^\omega(\mathcal{A}_\Phi) = [\![\Phi]\!]^\omega$, over *infinite* words.

[2]While a timed word θ is accepted by a timed automaton $\mathcal{B}$ if *there exists* an accepting run of $\mathcal{B}$ on θ, such a timed word is accepted by a *universal* timed automaton $\mathcal{A}$ if *all the runs* of $\mathcal{A}$ on θ are accepting.

To prove that TOCATA can be easily complemented, we first present another interesting property of TOCATA which concerns their acceptance condition. In the general case, a run of an OCATA is accepting iff all its branches visit accepting states infinitely often. Thanks to the partition characterising a TOCATA, this condition can be made simpler: a run of a TOCATA is accepting iff each branch eventually visits *only* accepting states (because it reaches a partition whose locations are all accepting).

Proposition 5.5. An f-run G_π of a TOCATA $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ is *accepting* iff $\forall \beta = \beta_0\beta_1 \ldots \beta_i \ldots \in Bran^\omega(G_\pi)$, $\exists n_\beta \in \mathbb{N}$ such that $\forall i > n_\beta$: $\beta_i = ((\ell, I), i)$ implies $\ell \in F$.

Proof. First, remark that the 'if' case is trivial. In the following, we prove the 'only if' case. As $\mathcal{A}$ is a tree-like OCATA, there exists a partition of L into disjoint subsets $L_1, L_2, \ldots, L_m$ satisfying:

1. $\forall 1 \leqslant i \leqslant m$ either $L_i \subseteq F$ or $L_i \cap F = \varnothing$, and
2. there is a partial order $\preccurlyeq$ on the sets $L_1, L_2, \ldots, L_m$ such that $L_j \preccurlyeq L_i$ iff $\exists \sigma \in \Sigma$, $\ell \in L_i$ and $\ell' \in L_j$ such that ℓ' is present in $\delta(\ell, \sigma)$.

Let π be an accepting f-run of $\mathcal{A}$ and G_π its associated DAG. Let $\beta = \beta_0\beta_1 \ldots \beta_i \ldots$ be a (finite or infinite) branch of G_π, we must prove that either β is finite, either $\exists n_\beta \in \mathbb{N}$ such that $\forall i > n_\beta$: $\beta_i = ((\ell, I), i)$ implies $\ell \in F$. As π is accepting, either β is finite, either there exists a smallest n_0 such that β_{n_0} has an accepting location. If β is finite, we are done. So, let us suppose that β is infinite. In this case, there exists a smallest n_0 such that β_{n_0} has an accepting location, say $\tilde{\ell} \in L_k$ for a certain $1 \leqslant k \leqslant m$. 1. implies that $L_k \subseteq F$. Thanks to 2., the location of β_{n_0+1} can only be:

a. a location of L_k, or

b. a location of L_j, for a certain $1 \leqslant j \leqslant m$ with $L_j \preccurlyeq L_k$ and $L_k \neq L_j$.

Remark that case a. can be repeated infinitely many times, while case b. can happen at most $m-1$ times. So, there exists $n^\star \geqslant n_0$ and $1 \leqslant i^\star \leqslant m$ such that $\forall i > n^\star$: $\beta_i = ((\ell, I), i)$ implies $\ell \in L_{i^\star}$. As π is an accepting f-run, 1. implies that $L_{i^\star} \subseteq F$ and $n^\star$ is the n_β we were looking for. □

We now define, from a TOCATA $\mathcal{A}$, the TOCATA $\mathcal{A}^C$ and prove that $L^\omega(\mathcal{A}^C) = T\Sigma^\omega \backslash L^\omega(\mathcal{A})$. The transition relation of $\mathcal{A}^C$ is obtained 'dualise' (see Definition 2.105) the transition relation of $\mathcal{A}$.

Definition 5.6. *For all TOCATA $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$, we let $\mathcal{A}^C = (\Sigma, L, \ell_0, L\backslash F, \overline{\delta})$ where $\overline{\delta}(\ell, \sigma) = \overline{\delta(\ell, \sigma)}$.*

Thanks to Proposition 5.5, we are able to prove that $\mathcal{A}^C$ accepts the complement of $\mathcal{A}$'s language. We base our proof on a key lemma claiming that, when considering a run G_π of $\mathcal{A}$ and a run $G_{\pi'}$ of $\mathcal{A}^C$ on a same timed word θ, G_π and $G_{\pi'}$ have a common infinite branch. We start by presenting the basis intuition of the proof in an example before to expose this lemma.

Example 5.7. Let $\mathcal{A}$ be a TOCATA and $\theta = (\overline{\sigma}, \overline{\tau})$ be an infinite timed word. Suppose that there exists a run π of $\mathcal{A}$ on θ and a run π' of $\mathcal{A}^C$ on θ. We here present an intuition of how a common beginning of branch of π and π' might be extended through one discrete transition. Let us so suppose that, after reading the i first letters of θ, π and π' reached configurations with a common state (ℓ, I). Let us furthermore suppose that:

$$\delta(\ell, \sigma_{i+1}) = (x \in I_1 \wedge \ell_1) \vee (x \in I_2 \wedge \ell_1 \wedge x.\ell_2).$$

Hence,

$$\overline{\delta}(\ell, \sigma_{i+1}) = (x \notin I_1 \vee \ell_1) \wedge (x \notin I_2 \vee \ell_1 \vee x.\ell_2).$$

We here explain why, after reading σ_{i+1}, the configurations reached by π and π' still contain a common state.
π is extended taking the transition $x \in I_1 \wedge \ell_1$ or the transition $x \in I_2 \wedge \ell_1 \wedge x.\ell_2$. Suppose for instance that π is constructed following the transition $x \in I_2 \wedge \ell_1 \wedge$

$x.\ell_2$: it will be extended in ℓ_1 and in ℓ_2 (with a reset of clock). π will so reach a configuration containing states (ℓ_1, I) and $(\ell_2, [0,0])$. Observing $\overline{\delta}(\ell_0, \sigma_1)$, we see we must choose an atom to satisfy among those of each transition of $\delta(\ell_0, \sigma_1)$. As π is constructed following transition $x \in I_2 \wedge \ell_1 \wedge x.\ell_2$, the clock constraint $x \in I_2$ is satisfied by all $j \in I$ and so $x \notin I_2$ will not be satisfied by any $j \in I$. Hence, every runs of $\mathcal{A}^C$ on θ will either be extended in ℓ_1 or in ℓ_2 (with a reset of clock). It means all these runs will reach a configuration containing either state (ℓ_1, I) or state $(\ell_2, [0,0])$. These possibly reached configurations will all have a common state with the configuration reached by π.

Lemma 5.8. *Let $\mathcal{A}$ be a TOCATA and θ be an infinite timed word. Let G_π be a run of $\mathcal{A}$ on θ and $G_{\pi'}$ be a run of $\mathcal{A}^C$ on θ. Then, G_π and $G_{\pi'}$ have a common infinite branch.*

Proof. Let $\mathcal{A}$ be a TOCATA and $\theta = (\sigma_1, \tau_1)(\sigma_2, \tau_2)(\sigma_3, \tau_3)....$ For all $i \geqslant 1$, we will note $t_i = \tau_i - \tau i - 1$, supposing $\tau_0 = 0$. Let π be a run of $\mathcal{A}$, denoted

$$C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \dots \overset{t_n}{\rightsquigarrow} C_{2n-1} \xrightarrow{\sigma_n} C_{2n} \dots,$$

whose DAG is $G_\pi = (V, \rightarrow)$ and let π' be a run of $\mathcal{A}^C$, denoted

$$C'_0 \overset{t_1}{\rightsquigarrow} C'_1 \xrightarrow{\sigma_1} C'_2 \overset{t_2}{\rightsquigarrow} C'_3 \xrightarrow{\sigma_2} \dots \overset{t_n}{\rightsquigarrow} C'_{2n-1} \xrightarrow{\sigma_n} C'_{2n} \dots,$$

whose DAG is $G_{\pi'} = (V', \dashrightarrow)$. We recall that $V = \bigcup_{0 \leqslant i \leqslant |\theta|} V_i$, where for all $0 \leqslant i \leqslant |\theta|$: $V_i = \{(\ell, i) \mid \ell \in C_i\}$ is the set of all vertices of depth i. In the same way, $V' = \bigcup_{0 \leqslant i \leqslant |\theta|} V'_i$, where for all $0 \leqslant i \leqslant |\theta|$: $V'_i = \{(\ell, i) \mid \ell \in C'_i\}$.
We will recursively construct a common infinite branch $\beta = \beta_0\beta_1\beta_2...$ of G_π and $G_{\pi'}$ such that for all $i \geqslant 0$, $\beta_i \in V_i \cap V'_i$.
<u>Basis:</u> $V_0 = V'_0 = \{((\ell_0, 0), 0)\}$, so, we take $\beta_0 = ((\ell_0, 0), 0)$.
<u>Induction:</u> Suppose we constructed a common beginning of branch of G_π and $G_{\pi'}$: $\beta_0\beta_1\beta_2...\beta_{n-1}$. Let us show that we can extend this common beginning of branch, constructing β_n. We want that $\beta_{n-1} \rightarrow \beta_n$ and $\beta_{n-1} \dashrightarrow \beta_n$.
Now, suppose that $C_{2n-2} = \{(\ell_i, v_i)_{i \in I}\}$, then $C_{2n-1} = \{(\ell_i, v_i + t_n)_{i \in I}\}$. Let $i^\star \in I$ such that $\beta_{n-1} = ((\ell_{i^\star}, v_{i^\star}), n-1)$. C_{2n} is obtained from C_{2n-1} thanks

to $\delta(\ell_i, \sigma_n)$ (for all $i \in I$), which can be written as (in disjunctive normal form) $\bigvee_{k \in K} a_k$ where a_k are atoms as described in Remarks 3.23 and 3.24. For a $v_i + t_n$ satisfying the guard present in a_k, the minimal models of $\delta(\ell_i, \sigma_n)$ with respect to $v_i + t_n$ are

$$\begin{aligned} a_k[v_i + t_n] &= \{(\ell, v_i + t_n) \mid \ell \text{ is a conjunct of } a_k\} \\ &\quad \cup \{(\ell, 0) \mid x.\ell \text{ is a conjunct of } a_k\}, \end{aligned}$$

$k \in K$. On the one hand, there exists a $k^\star \in K$ such that $v_{i^\star} + t_n$ satisfies the guard present in $a_{k^\star}$ and $a_{k^\star}[v_{i^\star} + t_n] \subseteq C_{2n}$: for each element (ℓ, v) of $a_{k^\star}[v_{i^\star} + t_n]$, $\beta_{n-1} \to ((\ell, v), n)$. On the other hand, let us observe the transition function $\overline{\delta}$ of $\mathcal{A}^C$ from $\ell_{i^\star}$: $\overline{\delta}(\ell_{i^\star}, \sigma_n) = \bigwedge_{k \in K} \overline{a_k}$. Let $k \in K$ and suppose $a_k = \ell_1 \wedge \cdots \wedge \ell_n \wedge x.(\ell_{n+1} \wedge \cdots \wedge \ell_m) \wedge \varphi$, where φ is a conjunction of clock constraints. Then, $\overline{a_k} = \ell_1 \vee \cdots \vee \ell_n \vee x.(\ell_{n+1} \vee \cdots \vee \ell_m) \vee \overline{\varphi_k}$: each clock constraint of φ must be negate to obtain $\overline{\varphi_k}$. As $v_{i^\star} + t_n$ satisfies all the clock constraints present in $a_{k^\star}$, it does not satisfy any clock constraint present in $\overline{a_{k^\star}}$. We know that $\beta_{n-1} = ((\ell_{i^\star}, v_{i^\star}), n-1) \in V'_{n-1}$, so that $(\ell_{i^\star}, v_{i^\star}) \in C'_{2n-2}$ and so $(\ell_{i^\star}, v_{i^\star} + t_n) \in C'_{2n-1}$. Each minimal model of $\overline{\delta}(\ell_{i^\star}, \sigma_n)$ with respect to $v_{i^\star} + t_n$ (which can be used to obtain C'_{2n}) must be such that, for each $k \in K$:

- either $v_{i^\star} + t_n$ satisfies a certain clock constraint present in $\overline{a_k}$,
- or this minimal model contains an element of $a_k[v_{i^\star} + t_n]$.

We showed that $v_{i^\star} + t_n$ does not satisfy any clock constraint present in $\overline{a_{k^\star}}$, so that the minimal model of $\overline{\delta}(\ell_{i^\star}, \sigma_n)$ with respect to $v_{i^\star} + t_n$ that is used in π' to construct C'_{2n} contains in particular an element $(\ell^\star, v^\star)$ of $a_{k^\star}[v_{i^\star} + t_n]$: $\beta_{n-1} \to ((\ell^\star, v^\star), n)$. As we previously showed that $\beta_n \to ((\ell, v), n)$ for each element (ℓ, v) of $a_{k^\star}[v_{i^\star} + t_n]$, we can choose $\beta_{n-1} = ((\ell^\star, v^\star), n)$. $\square$

We now prove that $\mathcal{A}^C$ accepts the complement of $\mathcal{A}$'s language, thanks to the previous lemma. The link between the transition relations δ and $\overline{\delta}$ highlighted in Example 5.7 still plays an important role in this proof.

Proposition 5.9. For all TOCATA $\mathcal{A}$, $L^\omega(\mathcal{A}^C) = T\Sigma^\omega \backslash L^\omega(\mathcal{A})$.

Proof. We will first prove that $L^\omega(\mathcal{A}) \cap L^\omega(\mathcal{A}^C) = \varnothing$, and then that $L^\omega(\mathcal{A}) \cup L^\omega(\mathcal{A}^C) = T\Sigma^\omega$.
$\underline{\mathbf{L}^\omega(\mathcal{A}) \cap \mathbf{L}^\omega(\mathcal{A}^\mathbf{C}) = \varnothing.}$ Suppose by contradiction that there exists a timed word θ such that $\theta \in L^\omega(\mathcal{A})$ and $\theta \in L^\omega(\mathcal{A}^C)$. Then, there exists accepting runs G_π and $G_{\pi'}$ (respectively) of $\mathcal{A}$ and $\mathcal{A}^C$ on θ. By Lemma 5.8, π and π' have a common infinite branch $\beta_0\beta_1\beta_2....$ As G_π is an accepting run of $\mathcal{A}$, by Property 5.5: $\exists n \in \mathbb{N}$, $\forall m > n$, the location of β_m is in F. But $G_{\pi'}$ is also an accepting run of $\mathcal{A}^C$, so that $\exists n' \in \mathbb{N}$, $\forall m > n$, the location of β_m is in $L \backslash F$: this is a contradiction.
$\underline{\mathbf{L}^\omega(\mathcal{A}) \cup \mathbf{L}^\omega(\mathcal{A}^\mathbf{C}) = \mathbf{T\Sigma}^\omega.}$ Let $\theta = (\sigma_1, \tau_1)(\sigma_2, \tau_2)(\sigma_3, \tau_3)...$ be a timed word such that $\theta \notin L^\omega(\mathcal{A})$, we must prove that $\theta \in L^\omega(\mathcal{A}^C)$. Let us note, for all $i \geqslant 1$, $t_i = \tau_i - \tau_{i-1}$, supposing $\tau_0 = 0$. We will inductively construct an accepting run of $\mathcal{A}^C$ on θ. To actually construct an *accepting* run, we will maintain an additional property of the beginning of run we extend at each inductive step, say π_{2n-2} denoted $C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \dots \overset{t_{n-1}}{\rightsquigarrow} C_{2n-3} \xrightarrow{\sigma_{n-1}} C_{2n-2}$, whose DAG is $G_{\pi_{2n-2}}$: 'for each branch β of $G_{\pi_{2n-2}}$, there exists a beginning of run π' of $\mathcal{A}$ on θ such that:

- either β is a branch of $G_{\pi'}$ and $G_{\pi'}$ cannot be extended into a complete run of $\mathcal{A}$ (it is 'blocking'),
- or π' can be extended into a complete run π'_c of $\mathcal{A}$ such that β is the beginning of a branch of G_{π_c} on which each location of F only occurs a finite number of times.'

We note $(^\star\mathbf{2n-2})$ this property.
<u>Basis.</u> We construct $C_0 = \{(\ell_0, 0)\}$. As there is no accepting run of $\mathcal{A}$ on θ, property $(^\star\mathbf{0})$ is trivially verified.
<u>Induction.</u> Suppose we constructed a beginning of run π_{2n-2} of $\mathcal{A}^C$ on θ: $C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \dots \overset{t_{n-1}}{\rightsquigarrow} C_{2n-3} \xrightarrow{\sigma_{n-1}} C_{2n-2}$ such that property $(^\star\mathbf{2n-2})$ is verified. We will extend π_{2n-2} to obtain a beginning of run $C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1}$

$C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \dots \overset{t_{n-1}}{\rightsquigarrow} C_{2n-3} \xrightarrow{\sigma_{n-1}} C_{2n-2} \overset{t_n}{\rightsquigarrow} C_{2n-1} \xrightarrow{\sigma_n} C_{2n}$ such that property (***2n**) is verified.

Suppose that $C_{2n-2} = \{(\ell_i, v_i)_{i\in I}\}$, then $C_{2n-1} = \{(\ell_i, v_i + t_n)_{i\in I}\}$. To form C_{2n}, each $(\ell_i, v_i + t_n) \in C_{2n-1}$ must evolve towards a minimal model of $\delta(\ell_i, \sigma_n)$ with respect to $v_i + t_n$. Let us note $\delta(\ell_i, \sigma_n)$ as $\bigvee_{k\in K} a_k$, as previously. As there is no accepting run of $\mathcal{A}$ on θ, whatever is the minimal model

$$\begin{aligned} a_k[v_i + t_n] \quad = \quad & \{(\ell, v_i + t_n) \mid \ell \text{ is a conjunct of } a_k\} \\ & \cup \{(\ell, 0) \mid x.\ell \text{ is a conjunct of } a_k\}, \end{aligned}$$

for $k \in K$, we decide to take as successor of $(\ell_i, v_i + t_n)$ in a beginning of run of $\mathcal{A}$ on θ, and no matter how it is then extended, it will contain a branch on which each location of F only occurs a finite number of times or will be 'blocking'. Let us note $(\ell_i^k, v_i^k)_{i\in I, k\in K}$ the successors of $(\ell_i, v_i + t_n)$ on the branches on which each location of F only occurs a finite number of timed or are 'blocking'[3], considering a certain run of $\mathcal{A}$ on θ going from $(\ell_i, v_i + t_n)$ to the minimal model $a_k[v_i + t_n]$. We construct $C_{2n+2} = \{(\ell_i^k, v_i^k)_{i\in I, k\in K}\}$ (it is actually the union, for $i \in I$, of minimal models of $\overline{\delta}(\ell_i, \sigma_n) = \bigwedge_{k\in K} \overline{a_k}$ with respect to $v_i + t_n$). By construction, (***2n**) is verified.

Thanks to this induction, we can construct an infinite run π, $C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \dots \overset{t_n}{\rightsquigarrow} C_{2n-1} \xrightarrow{\sigma_n} C_{2n} \dots$, such that for each branch β of G_π, there exists a run or a beginning of run π' of $\mathcal{A}$ on θ such that:

- either β is a branch of $G_{\pi'}$ and $G_{\pi'}$ cannot be extended into a complete run of $\mathcal{A}$ (it is 'blocking'): it means that β is a finite and non-blocking branch of G_π,

[3]Remark that a branch of a run of $\mathcal{A}$ is blocking reading σ_n iff a_k contains a clock constraint $x \bowtie c$ not satisfied in $v_i + t_n$ or there is no transition from ℓ_i reading σ_n. In the first case $\overline{a_k}$ contains the negation of $x \bowtie c$, which is verified by $v_i + t_n$, and in the second case $\delta(\ell_i, \sigma_n)$ is *false* and so $\overline{\delta}(\ell_i, \sigma_n)$ is *true*. In both cases (in particular) $\overline{a_k}$ will be satisfied in $\overline{\delta}(\ell_i, \sigma_n)$, even if no successor is attributed to $(\ell_i, v_i + t_n)$ (the branch ending in $(\ell_i, v_i + t_n)$ in π_{2n-1} is finite but not blocking).

- or π' is a run of $\mathcal{A}$ containing the branch β, on which each location of F only occurs a finite number of times.

So, all the infinite branches of π visit $L \backslash F$ infinitely often and π is an accepting run of $\mathcal{A}^C$ on θ. □

Equipped with those results, we can now set out the main result of this section: the translation from MTL to OCATA introduced in [51] carries on to infinite words. To the best of our knowledge this had not been proved before and does not seem completely trivial since our proof requires the machinery of TOCATA developed in this thesis.

This is in way to prove this main result that we needed Proposition 5.9, showing that TOCATA are easily complementable, contrary to general OCATA.

Theorem 5.10. *For every MTL formula Φ: $L^\omega(\mathcal{A}_\Phi) = [\![\Phi]\!]^\omega$.*

Proof. This has been proved in the finite words case in [51, Prop. 6.4]. This proof relies crucially on the fact that OCATA can be complemented in this setting. Thanks to Proposition 5.9, we can trivially adapt the proof of [51]. □

5.2 From MITL to Büchi Timed Automata

In this section, we present our new technique to build, from any MITL formula Φ, a *Büchi timed automaton* that accepts $[\![\Phi]\!]^\omega$. Our construction relies on two ingredients. The first one is still the *bounded approximation functions* $f^\star_\Phi$. The second ingredient is the adaptation of the Miyano Hayashi construction (see Example 2.48) in a timed context.

When considering an MITL formula Φ, interpreted over infinite words, the bounded approximation function $f^\star_\Phi$ still enables to bound the number of clock

copies needed in the $f^\star_\Phi$-semantics of $\mathcal{A}_\Phi$, as stated by the following theorem. Its proof is based on Propositions 4.8 and 4.9, which are invariably true over finite or infinite words: their proofs can be found in the appendix. The body of the proof of this theorem is identical to that of Theorem 4.7 (its twin over finite words). Indeed, the bound on the number of clock copies needed in $\mathcal{A}_\Phi$ depends only on the MITL formula Φ, not on the way the run propagates.

Theorem 5.11. *For all MITL formula Φ, $f^\star_\Phi$ is a* bounded approximation function *and $L^\omega_{f^\star_\Phi}(\mathcal{A}_\Phi) = L^\omega(\mathcal{A}_\Phi) = [\![\Phi]\!]^\omega$.*

As for the finite words case, this bound will enable to construct a Büchi timed automaton with a finite number of locations and clocks.

When we presented the transition from the OCATA $\mathcal{A}_\Phi$ to the timed automaton $\mathcal{B}_\Phi$ over *finite* words, to manage the acceptance condition of the constructed $\mathcal{B}_\Phi$ was not complex: it sufficed to choose the adapted accepting locations. In the setting of *infinite* words, to manage the acceptance condition will not be so simple: it will require to use the Miyano-Hayashi method. Their approach consists in associating a marker ($\top$ or $\bot$) to each state of each configuration of the OCATA. In way to overtake this difficulty, we start by defining, by means of such markers, a timed transition system $\mathsf{MHTS}\,(\mathcal{A}, f)$ (from an OCATA $\mathcal{A}$ and an approcimation function f) with a *Büchi acceptance condition*, recognizing $L_f(\mathcal{A})$. Once the difficulty linked to the introduction of Miyano-Hayashi markers overtaken, we present the construction of a Büchi timed automaton $\mathcal{B}_\Phi$ accepting $[\![\Phi]\!]^\omega$. Its construction will be similar to that presented over finite words thanks to the use of $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f^\star_\Phi)$.

5.2.1 A Büchi transition system for each OCATA

The translation from an OCATA to a Büchi timed automaton is not trivial. For the untimed case yet, this translation is obtained by Miyano and Hayashi using a trick consisting in associating a marker to each state of each configuration

of the alternating automaton. The use of such markers enable to get back a Büchi acceptance condition. In this section, we adapt their approach to the timed setting in way to obtain a timed transition system $\mathsf{MHTS}\,(\mathcal{A}, f)$ with a *Büchi acceptance condition*, recognizing $L_f(\mathcal{A})$.

As explained in Example 2.48, the Miyano-Hayashi method will associate a marker, $\top$ or $\bot$, to each state of each configuration of the OCATA $\mathcal{A}$. Intuitively, a state is marked by $\top$ if all the branches it belongs to have visited an accepting location of $\mathcal{A}$. When a configuration C whose states are all marked by $\top$ is reached, it means that all the branches leading to C saw F: such configurations will be *accepting states* of $\mathsf{MHTS}\,(\mathcal{A}, f)$. When such an accepting configuration is reached, all its states are marked back to $\bot$ and the marking to $\top$ starts again. Hence, a state of a configuration is marked by $\top$ iff all the branches it belongs to have visited an accepting location of $\mathcal{A}$ *since the last accepting state* of $\mathsf{MHTS}\,(\mathcal{A}, f)$ was reached. We so recover a Büchi acceptance condition in the following way: each branch of a run of $\mathcal{A}$ see F infinitely often (i.e it is an accepting run of $\mathcal{A}$) iff we encounter infinitely many accepting states of $\mathsf{MHTS}\,(\mathcal{A}, f)$ in the corresponding run of $\mathsf{MHTS}\,(\mathcal{A}, f)$.

Here is the formal definition of $\mathsf{MHTS}\,(\mathcal{A}, f) = (\Sigma, S^{\mathsf{MH}}, s_0^{\mathsf{MH}}, \rightsquigarrow^{\mathsf{MH}}, \rightarrow^{\mathsf{MH}}, \alpha)$. The states of $\mathsf{MHTS}\,(\mathcal{A}, f)$ will be *marked* configurations of $\mathcal{A}$, i.e. sets of trios of the form (ℓ, I, m), where (ℓ, I) is a state of $\mathcal{A}$ and $m \in \{\top, \bot\}$. The discrete transition relation, represented by '$\rightarrow^{\mathsf{MH}}$', will be defined in way that (i) a transition with $\rightarrow^{\mathsf{MH}}$ between two states of S^{MH} corresponds to a discrete transition between the configurations of the OCATA $\mathcal{A}$ they represent and (2) the markers of the third components are kept updated. Actually, if $s \xrightarrow{\sigma}^{\mathsf{MH}} s'$, we want the last component of a trio of s' to be $\bot$ iff its first component is not in F and it comes from the grouping of trios emanating from trios of s such that at least one of them had $\bot$ as last component. Moreover, if $s \in \alpha$, we want to start again the marking: we put all the third components of the trios of s to $\bot$ before proceeding as previously.

Definition 5.12. *For an OCATA $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ and an approximation*

function f, we define the timed transition system $\mathsf{MHTS}\,(\mathcal{A}, f) = (\Sigma, S^{\mathsf{MH}}, s_0^{\mathsf{MH}}, \rightsquigarrow^{\mathsf{MH}}, \rightarrow^{\mathsf{MH}}, \alpha)$ *where:*

(i) $S^{\mathsf{MH}} = \{(\ell_k, I_k, m_k)_{k\in K} \mid \{(\ell_k, I_k)_{k\in K}$ *is a configuration of* $\mathcal{A}$ *and* $\forall k \in K$, $m_k \in \{\top, \bot\}\}$,

(ii) $s_0^{\mathsf{MH}} = \{(\ell_0, [0,0], m)\}$, *with* $m = \top$ *iff* $\ell_0 \in F$,

(iii) $\alpha = \{(\ell_k, I_k, m_k)_{k\in K} \in S^{\mathsf{MH}} \mid \forall k \in K, m_k = \top\}$,

(iv) for $t \in \mathbb{R}$ *and* $s, s' \in S$, *supposing* $s = \{(\ell_k, I_k, m_k)_{k\in K}\}$, *we have* $s \overset{t}{\rightsquigarrow}^{\mathsf{MH}} s'$ *iff* $s' = \{(\ell_k, I_k + t, m_k)_{k\in K}\}$; $\rightsquigarrow^{\mathsf{MH}} = \bigcup_{t\in\mathbb{R}} \overset{t}{\rightsquigarrow}^{\mathsf{MH}}$,

(v) for $s \in S^{\mathsf{MH}} \backslash \alpha$ *and* $s' \in S^{\mathsf{MH}}$, *supposing* $s = \{(\ell_k, I_k, m_k)_{k\in K}\}$ *and* $s' = \{(\ell_{k'}, I_{k'}, m_{k'})_{k'\in K'}\}$, $s \overset{\sigma}{\rightarrow}^{\mathsf{MH}} s'$ *iff*

(a) $\{(\ell_k, I_k)_{k\in K}\} \overset{\sigma}{\rightarrow}_f \{(\ell_{k'}, I_{k'})_{k'\in K'}\}$ *in* $\mathcal{A}$, *i.e.*

$$\{(\ell_{k'}, I_{k'})_{k'\in K'}\} \in f(\mathsf{Succ}(\{(\ell_k, I_k)_{k\in K}\}, \sigma));$$

(b) $\forall k' \in K'$: $(\ell_{k'} \in F \;\Rightarrow\; m_{k'} = \top)$;

(c) $\forall \overline{k} \in K'$ *with* $\ell_{\overline{k}} \notin F$: *if* $\exists k^* \in K$ *such that* $(\ell_{k^*}, I_{k^*}, \bot) \in s$ *and* $(\ell_{\overline{k}}, I_{\overline{k}}) \in \mathsf{dest}(\{(\ell_k, I_k)_{k\in K}\}, \{(\ell_{k'}, I_{k'})_{k'\in K'}\}, (\ell_{k^*}, I_{k^*}))$, *we have* $m_{\overline{k}} = \bot$; *otherwise,* $m_{\overline{k}} = \top$,

(vi) for $s \in \alpha$ *and* $s' \in S^{\mathsf{MH}}$, *supposing* $s = \{(\ell_k, I_k, \top)_{k\in K}\}$, $s \overset{\sigma}{\rightarrow}^{\mathsf{MH}} s'$ *iff* $\{(\ell_k, I_k, \bot)_{k\in K}\} \overset{\sigma}{\rightarrow}^{\mathsf{MH}} s'$ *according to the rules in (v)* ;
$\rightarrow^{\mathsf{MH}} = \bigcup_{\sigma\in\Sigma} \overset{\sigma}{\rightarrow}^{\mathsf{MH}}$,

We will now prove that, in general, the language of the Büchi transition system $\mathsf{MHTS}\,(\mathcal{A}, f)$ is $L_f^\omega(\mathcal{A})$. In particular, it will mean that $L^\omega(\mathsf{MHTS}\,(\mathcal{A}_\Phi, f_\Phi^\star)) = L_{f_\Phi^\star}^\omega(\mathcal{A}_\Phi)$. This will enable us to define a Büchi timed automaton $\mathcal{B}_\Phi$ recognizing $[\![\Phi]\!]^\omega$ $(= L_{f_\Phi^\star}^\omega(\mathcal{A}_\Phi))$ using $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f_\Phi^\star)$.

Proposition 5.13. Let $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ be an OCATA and f be an approximation function: $L^\omega(\mathsf{MHTS}\,(\mathcal{A}, f)) = L_f^\omega(\mathcal{A})$.

Proof. ($\supseteq$) Let $\theta = (\overline{\sigma}, \overline{\tau}) \in L_f^\omega(\mathcal{A})$, with $\overline{\sigma} = \sigma_1\sigma_2\cdots\sigma_n\ldots$ and $\overline{\tau} = \tau_1\tau_2\cdots\tau_n\ldots$. We will prove that $\theta \in L^\omega(\mathsf{MHTS}\,(\mathcal{A}, f))$. Let us note $t_i = \tau_i - \tau_{i-1}$ for all $1 \leqslant i \leqslant |\theta|$, assuming $\tau_0 = 0$. We have an accepting f-run of $\mathcal{A}$ on θ, say $\pi : C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1}_f C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2}_f \ldots \overset{t_i}{\rightsquigarrow} C_{2i-1} \xrightarrow{\sigma_i}_f C_{2i}\ldots$. We must prove that there is an accepting run of $\mathsf{MHTS}\,(\mathcal{A}, f)$ on θ, say π': $s_0 \overset{t_1}{\rightsquigarrow}^{\mathsf{MH}} s_1 \xrightarrow{\sigma_1}^{\mathsf{MH}} s_2 \overset{t_2}{\rightsquigarrow}^{\mathsf{MH}} s_3 \xrightarrow{\sigma_2}^{\mathsf{MH}} \ldots \overset{t_i}{\rightsquigarrow}^{\mathsf{MH}} s_{2i-1} \xrightarrow{\sigma_i}^{\mathsf{MH}} s_i\ldots$. We construct π' by induction, proving additionally that the two following properties hold for $j \geqslant 0$:

($\star 2j$) $C_{2j} = \{(\ell_k, I_k)_{k\in K}\}$ iff $s_{2j} = \{(\ell_k, I_k, m_k)_{k\in K} \mid m_k \in \{\top, \bot\}\}$;

($*2j$) if a location of F occurs on all the branches of π between the last configuration $C_{2j'}$ such that $s_{2j'} \in \alpha$ (or, failing that, between C_0) and C_{2j}, then, $s_{2j} \in \alpha$.

<u>Basis:</u> We know that $C_0 = (\ell_0, [0, 0])$ and $s_0 = \{(\ell_0, [0, 0], \bot)\}$. It is clear that ($\star 0$) and ($*0$) are verified because only $\ell_0 \notin F$ occurs on the (unique) branch of π.

<u>Induction:</u> Suppose that we constructed π' until s_{2i} and that ($\star 2j$) and ($*2j$) are verified $\forall 0 \leqslant j \leqslant i$. We will construct π' until $s_{2(i+1)}$ in way ($\star 2(i+1)$) and ($*2(i+1)$) will still be verified.

First, we must construct s_{2i+1} such that $s_{2i} \overset{t_{i+1}}{\rightsquigarrow}^{\mathsf{MH}} s_{2i+1}$. Suppose $s_{2i} = \{(\ell_k, I_k, m_k)_{k\in K}\}$, as ($\star 2i$) is verified by hypothesis, it means that C_{2i} can also be written as $\{(\ell_k, I_k)_{k\in K}\}$. We must choose s_{2i+1} to be $\{(\ell_k, I_k + t_{i+1}, m_k)_{k\in K}\}$. As $C_{2i} \overset{t_{i+1}}{\rightsquigarrow} C_{2i+1}$, $C_{2i+1} = C_{2i} + t_{i+1} = \{(\ell_k, I_k + t_{i+1})_{k\in K}\}$. Remark that the following property holds:

($\star 2i+1$) $\quad C_{2i+1} = \{(\ell_k, I_k)_{k\in K}\}$ iff $s_{2i+1} = \{(\ell_k, I_k, m_k)_{k\in K} \mid m_k \in \{\top, \bot\}\}$.

Secondly, we must construct $s_{2(i+1)}$ such that $s_{2i+1} \xrightarrow{\sigma_{i+1}} s_{2(i+1)}^{\mathsf{MH}}$. We know that $C_{2i+1} \xrightarrow{\sigma_{i+1}}_f C_{2(i+1)}$. Suppose $s_{2i+1} = \{(\ell_k, I_k, m_k)_{k\in K}\}$, as ($\star 2i+1$) is verified, C_{2i+1} can be written as $\{(\ell_k, I_k)_{k\in K}\}$. Let us further suppose that

$C_{2(i+1)} = \{(\ell_{k'}, I_{k'})_{k' \in K'}\}$. We construct $s_{2(i+1)} := \{(\ell_{k'}, I_{k'}, m_{k'})_{k' \in K'}\}$ to be the unique state[4] of S^{MH} such that $\{(\ell_k, I_k)_{k \in K}\} \xrightarrow{\sigma_{i+1}}_f \{(\ell_{k'}, I_{k'})_{k' \in K'}\}$. $(\star 2(i+1))$ is trivially verified. It remains to prove that $(*2(i+1))$ is satisfied. Suppose that a location of F occurs on all the branches of π between the last configuration $C_{2j'}$ such that $s_{2j'} \in \alpha$ (or, failing that, between C_0) and $C_{2(i+1)}$. We must prove that $s_{2(i+1)} \in \alpha$, i.e. all the trios of $s_{2(i+1)}$ has $\top$ as last component. Let $\beta = \beta_0\beta_1\beta_2 \ldots \beta_{2j'} \ldots \beta_{2j} \ldots \beta_{2(i+1)} \ldots$ be a branch of π. The hypothesis implies there exists a transition $\xrightarrow{\sigma_j}_f$, for $j' \leqslant j \leqslant i+1$, such that $\beta_{2j} = (\ell, I)$ for a certain $\ell \in F$. By $(*2j)$, $(\ell, I, m_k) \in s_{2j}$ for a certain m_k in $\{\top, \bot\}$, but point (v)-(b) of the definition of $\mathsf{MHTS}\,(\mathcal{A}, f)$ (Definition 5.12) obliges m_k to be $\top$. So, the third components associated in π' to the different states of the branches of π will gradually (between $s_{2j'}$ and $s_{2(i+1)}$) become $\top$. We must still ensure they will eventually never become $\bot$ again. In fact, a pair (ℓ, I) of a certain state s_j (for $2j' \leqslant j \leqslant 2(i+1)$) of π, corresponding to $(\ell, I, \top)$ in π', can have a successor (ℓ', I') in π and a corresponding successor $(\ell', I', \bot)$ in π' *iff* s_j is accepting and $\ell' \notin F$ (which is not possible under the present hypothesis) or $(\ell', I', \bot)$ comes from the grouping of trios (thanks to f) emanating from trios such that at least one of them had $\bot$ as last component (case (v)-(c) of the definition of $\mathsf{MHTS}\,(\mathcal{A}, f)$: Definition 5.12). It means that no location of F occurs on one of the branches of π leading to (ℓ', I'), say $\beta' = \beta'_0\beta'_1\beta'_2 \ldots \beta_{2j'} \ldots \beta'_{2k'} \ldots \beta'_{2(i+1)} \ldots$, with $\beta'_{2k} = (\ell', I')$, since $\beta'_{2j'}$ (else, we contradict case (v)-(c) of the definition of $\mathsf{MHTS}\,(\mathcal{A}, f)$: Definition 5.12). But, by hypothesis, a location of F occurs on all the branches of π between steps $2j'$ and $2(i+1)$, so there exists a transition $\xrightarrow{\sigma_{2\tilde{j}}}_f$, for $k' < \tilde{j} \leqslant i+1$, such that $\beta'_{2\tilde{j}}$ has its location in F: once again, point (v)-(b) of the definition of $\mathsf{MHTS}\,(\mathcal{A}, f)$ (Definition 5.12) obliges $m_{2\tilde{j}}$ to be $\top$.
As a location of F occurs on all branches of π between steps $2j'$ and $2(i+1)$ and there is only a finite number of branches leading to a state of $C_{2(i+1)}$, we conclude that we can only encounter this last case a finite number of times and so $s_{2(i+1)} \in \alpha$: $(*2(i+1))$ is satisfied.

[4] once the minimal models of the definition of $\xrightarrow{\sigma_{i+1}}$ are chosen, it is easy to see there exists a *unique* possible choice for the values of the m_k of $s_{2(i+1)}$.

To end this part of the proof, we must show that π' is accepting. The previous induction proves that $(*2j)$ is verified for all $j \geqslant 0$. As π is accepting, a location of F occurs on all the branches of π infinitely often, and so there is an infinite number of j and j' such that the antecedent of $(*2j)$ is true. So, in this same infinite number of times, we know that $s_{2j} \in \alpha$, what proves that π' is accepting.
($\subseteq$) Let $\theta = (\overline{\sigma}, \overline{\tau}) \in L^\omega(\mathsf{MHTS}\,(\mathcal{A}, f))$, with $\overline{\sigma} = \sigma_1\sigma_2\cdots\sigma_n\ldots$ and $\overline{\tau} = \tau_1\tau_2\cdots\tau_n\ldots$. We will prove that $\theta \in L^\omega_f(\mathcal{A})$. Let us note $t_i = \tau_i - \tau_{i-1}$ for all $1 \leqslant i \leqslant |\theta|$, assuming $\tau_0 = 0$. We have an accepting run of $\mathsf{MHTS}\,(\mathcal{A}, f)$ on θ, say π: $s_0 \overset{t_1}{\rightsquigarrow}^{\mathsf{MH}} s_1 \overset{\sigma_1}{\longrightarrow}^{\mathsf{MH}} s_2 \overset{t_2}{\rightsquigarrow}^{\mathsf{MH}} s_3 \overset{\sigma_2}{\longrightarrow}^{\mathsf{MH}} \ldots \overset{t_i}{\rightsquigarrow}^{\mathsf{MH}} s_{2i-1} \overset{\sigma_i}{\longrightarrow}^{\mathsf{MH}} s_i \ldots$. We must prove that there is an accepting f-run of $\mathcal{A}$ on θ, say π': $C_0 \overset{t_1}{\rightsquigarrow} C_1 \overset{\sigma_1}{\longrightarrow}_f C_2 \overset{t_2}{\rightsquigarrow} C_3 \overset{\sigma_2}{\longrightarrow}_f \ldots \overset{t_i}{\rightsquigarrow} C_{2i-1} \overset{\sigma_i}{\longrightarrow}_f C_{2i} \ldots$. We construct π' by induction, proving additionally that the two following properties hold for $j \geqslant 0$:

$(\star 2j)$ $C_{2j} = \{(\ell_k, I_k)_{k\in K}\}$ iff $s_{2j} = \{(\ell_k, I_k, m_k)_{k\in K} \mid m_k \in \{\top, \bot\}\}$;

$(*2j)$ if $s_{2j} \in \alpha$, then, a location of F occurs on all the branches of π' between the last configuration $C_{2j'}$ such that $s_{2j'} \in \alpha$ (or, failing that, between C_0) and C_{2j}.

<u>Basis:</u> We know that $s_0 = \{(\ell_0, [0,0], \bot)\}$ and $C_0 = (\ell_0, [0,0])$. $(\star 0)$ and $(*0)$ are trivially verified.
<u>Induction:</u> Suppose that we constructed π' until C_{2i} and that $(\star 2j)$ and $(*2j)$ are verified $\forall 0 \leqslant j \leqslant i$. We will construct π' until $C_{2(i+1)}$ in way $(\star 2(i+1))$ and $(*2(i+1))$ will still hold.
First, we must construct C_{2i+1} such that $C_{2i} \overset{t_{i+1}}{\rightsquigarrow} C_{2i+1}$. We know that $s_{2i} \overset{t_{i+1}}{\rightsquigarrow}^{\mathsf{MH}} s_{2i+1}$. Suppose $s_{2i} = \{(\ell_k, I_k, m_k)_{k\in K}\}$, as $(\star 2i)$ is verified by hypothesis, $C_{2i} = \{(\ell_k, I_k)_{k\in K}\}$. We must choose C_{2i+1} to be $C_{2i+1} = \{(\ell_k, I_k + t_{i+1})_{k\in K}\}$. As $s_{2i} \overset{t_{i+1}}{\rightsquigarrow}^{\mathsf{MH}} s_{2i+1}$, $s_{2i+1} = \{(\ell_k, I_k + t_{i+1}, m_k)_{k\in K}\}$. Remark that the following property holds:

$(\star 2i+1)$ $\quad C_{2i+1} = \{(\ell_k, I_k)_{k\in K}\}$ iff $s_{2i+1} = \{(\ell_k, I_k, m_k)_{k\in K} \mid m_k \in \{\top, \bot\}\}$.

Secondly, we must construct $C_{2(i+1)}$ such that $C_{2i+1} \xrightarrow{\sigma_{i+1}}_f C_{2(i+1)}$. We know that $s_{2i+1} \xrightarrow{\sigma_{i+1}}^{\mathsf{MH}} s_{2(i+1)}$. Suppose $s_{2i+1} = \{(\ell_k, I_k, m_k)_{k\in K}\}$, as $(\star 2i+1)$ is verified, $C_{2i+1} = \{(\ell_k, I_k)_{k\in K}\}$. Let us further suppose that $s_{2i+1} = \{(\ell_{k'}, I_{k'}, m_{k'})_{k'\in K'}\}$. We construct $C_{2(i+1)} = \{(\ell_{k'}, I_{k'})_{k'\in K'}\}$, so that $(\star 2(i+1))$ holds. Remark that, by definition of $\rightarrow^{\mathsf{MH}}$: $s_{2i+1} \xrightarrow{\sigma_{i+1}}^{\mathsf{MH}} s_{2(i+1)}$ and so we have $C_{2i+1} \xrightarrow{\sigma_{i+1}}_f C_{2(i+1)}$ (what we needed). It remains to prove that $(*2(i+1))$ is satisfied. Suppose that $s_{2(i+1)} \in \alpha$ (i.e. $\forall k' \in K'$, $m_{k'} = \top$). We must prove that, between the last configuration $C_{2j'}$ such that $s_{2j'} \in \alpha$ (or, failing that, between C_0) and $C_{2(i+1)}$, a location of F occurs on all the branches of π'. Let $\beta = \beta_0\beta_1\beta_2\ldots\beta_{2j'}\ldots\beta_{2j}\ldots\beta_{2(i+1)}\ldots$ be a branch of π' and suppose by contradiction that $\forall j' \leqslant j \leqslant i+1$, β_{2j} has not its location in F. As $s_{2j'} \in \alpha$, all the third components of $s_{2j'}$ are replaced by $\perp$ (likewise, by definition of s_0, its unique trio has $\perp$ as last component) before evolving reading $\sigma_{j'+1}, \sigma_{j'+2}, \ldots, \sigma_{i+1}$ thanks to the rules in (v) in the definition of $\mathsf{MHTS}\,(\mathcal{A}, f)$ (Definition 5.12). But, observing those rules in (v), when a trio has $\perp$ as last component, it can only evolve to a trio with $\top$ as last component if case (c) is satisfied, what is impossible along β. This contradicts the fact that $s_{2(i+1)} \in \alpha$.

To end the proof, we must show that π' is accepting. The previous induction proves that $(*2j)$ holds for all $j \geqslant 0$. As π is accepting, we know that $s_{2j} \in \alpha$ for infinitely many j and so, between any two successive such j's all the branches of π' saw F. We conclude that all the branches of π' saw F infinitely often, what proves that π' is accepting. □

5.2.2 Towards a Büchi timed automaton

The aim of this subsection is to define a timed automaton with Büchi acceptance condition $\mathcal{B}_\Phi$ that simulates $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f^\star_\Phi)$, and thus accepting $[\![\Phi]\!]^\omega$, for every MITL formula Φ. For the sake of simplicity, we uses the same notations as for the finite words case. We so note $\mathcal{B}_\Phi = (\Sigma, L, \lambda_0, X, F, \delta)$, and still denote $\mathsf{loc}(X)$ the set of functions representing the locations of $\mathcal{B}_\Phi$ over infinite words, although the definitions of $\mathsf{loc}(X)$ and of the components of $\mathcal{B}_\Phi$ differ from the

finite word case. We hope this will help the reader to make a parallel between our definitions of (Büchi) timed automata $\mathcal{B}_\Phi$, accepting $[\![\Phi]\!]$ over finite words and $[\![\Phi]\!]^\omega$ over infinite words.

Let Φ be an MITL formula, and assume $\mathcal{A}_\Phi = (\Sigma, L^\Phi, \ell_0^\Phi, F^\Phi, \delta^\Phi)$ is its associated OCATA (see Definition 2.164). Let us show how to build a TA $\mathcal{B}_\Phi = (\Sigma, L, \lambda_0, X, F, \delta)$ such that $L^\omega(\mathcal{B}_\Phi) = L^\omega(\mathsf{MHTS}\,(\mathcal{A}_\Phi, f_\Phi^\star))$. For the sake of simplicity, in the definition of $\mathcal{B}_\Phi$, we consider that all the intervals used by $\mathcal{A}_\Phi$, even the singular ones, are represented by two different clock copies.
We recall that, in the finite word settings, the locations of $\mathcal{B}_\Phi$ (see Definition 4.13) are functions $\lambda : L^\Phi \to 2^{(X^2)}$ satisfying the property that each clock is at most present one time among those in $\bigcup_{\ell \in L^\Phi} \lambda(\ell)$. For infinite words, this definition must be reviewed to contain Miyano-Hayashi markers, as in $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f_\Phi^\star)$. A location of $\mathcal{B}_\Phi$ will now be a function $\lambda : L^\Phi \to 2^{(X^2 \times \{\top, \bot\})}$, still satisfying the property that each clock is at most present one time among those in $\bigcup_{\ell \in L^\Phi} \lambda(\ell)$. Hereunder, we formally define the set of functions enabling to represent locations of $\mathcal{B}_\Phi$, over infinite words. Then, we present the definition of the first components of $\mathcal{B}_\Phi$: its transition function, more complex to define, will be detailed right after.

Definition 5.14. *We define* $\mathsf{loc}(X)$ *to be set of functions* $\lambda : L^\Phi \to 2^{(X^2 \times \{\top, \bot\})}$ *such that, for all* $(x, y, m) \in \bigcup_{\ell \in L^\Phi} \lambda(\ell)$*, we have that* $x \neq y$ *and, for all* $(x', y', m') \in (\bigcup_{\ell \in L^\Phi} \lambda(\ell)) \setminus \{(x, y, m)\}$*, we have that* $x \neq x'$*,* $x \neq y'$*,* $y \neq x'$ *and* $y \neq y'$*.*

As for the finite words case, thanks to the bound $M(\Phi)$ of clock copies needed in the OCATA $\mathcal{A}_\Phi$ (given by Theorem 5.11), we will only need a set X of $M(\Phi)$ clocks in $\mathcal{B}_\Phi$. This way, $\mathsf{loc}(X)$ is a finite set of functions, which will play the role of locations of $\mathcal{B}_\Phi$. Without the bound given by Theorem 5.11, $\mathcal{B}_\Phi$ would have had an infinite number of locations and would not have been a timed automaton.

Definition 5.15. *We define* $\mathcal{B}_\Phi = (\Sigma, L, \lambda_0, X, F, \delta)$*, where:*

- X *is a set of clocks such that* $|X| = M(\Phi)$*,*
- $L = \mathsf{loc}(X)$*,*

- $\lambda_0 \in \mathsf{loc}$ *is such that* $\lambda_0(\ell_0^\Phi) = \{(x, y, \bot)\}$, *where* x *and* y *are two clocks arbitrarily chosen from* X, *and* $\lambda_0(\ell) = \varnothing$ *for all* $\ell \in L^\Phi \backslash \{\ell_0^\Phi\}$.
- F *is the set of all locations* $\lambda \in L$ *such that:* $\forall \ell \in L^\Phi$, $\forall (x, y, m) \in \lambda(\ell)$, $m = \top$.

Intuitively, a configuration (λ, v) of $\mathcal{B}_\Phi$ encodes the state s of $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f_\Phi^\star)$ such that: $s = \{(\ell, [v(x), v(y)], m) \mid \ell \in L^\Phi$ and $(x, y, m) \in \lambda(\ell)\}$, i.e. the marked intervals associated to location ℓ are given by the marker and values (according to v) of pairs of clocks in $\lambda(\ell)$.
Finally, we must define the set of transitions δ to let $\mathcal{B}_\Phi$ simulate the executions of $\mathcal{A}_\Phi$. We recall (see Remark 3.24) that for each location $\ell \in L^\Phi$, for each $\sigma \in \Sigma$, all *arcs* in δ^Φ are either of the form $(\ell, \sigma, \mathit{true})$, or $(\ell, \sigma, \mathit{false})$, or $\big(\ell, \sigma, \ell \wedge x.(\ell_1 \wedge \cdots \wedge \ell_k) \wedge g\big)$ or of the form $\big(\ell, \sigma, x.(\ell_1 \wedge \cdots \wedge \ell_k) \wedge g\big)$, where g is a *guard* on x. Let $\lambda \in L$ be a location of $\mathcal{B}_\Phi$, $\ell \in L^\Phi$, $\sigma \in \Sigma$ be a letter and (x, y, m) be a trio occurring in $\lambda(\ell)$. Let us associate to this trio an arc a of δ^Φ of the form (ℓ, σ, γ). Then, as in the finite words setting, we associate to a a *guard* $\mathsf{guard}\,(a)$, and two sets $\mathsf{reset}\,(a)$ and $\mathsf{loop}\,(a)$, defined as follows:

- if $\gamma \in \{\mathit{true}, \mathit{false}\}$, then, $\mathsf{guard}\,(a) = \gamma$ and $\mathsf{reset}\,(a) = \mathsf{loop}\,(a) = \varnothing$.
- if γ is of the form $x.(\ell_1 \wedge \cdots \wedge \ell_k) \wedge g$, then $\mathsf{guard}\,(a) = g$, $\mathsf{reset}\,(a) = \{\ell_1, \ldots, \ell_k\}$ and $\mathsf{loop}\,(a) = \varnothing$.
- if γ is of the form $\ell \wedge x.(\ell_1 \wedge \cdots \wedge \ell_k) \wedge g$, then $\mathsf{guard}\,(a) = g$, $\mathsf{reset}\,(a) = \{\ell_1, \ldots, \ell_k\}$ and $\mathsf{loop}\,(a) = \{(x, y)\}$.

Thanks to those definitions, we can now define δ. Let $\lambda^\triangle \in L$ be a location of $\mathcal{B}_\Phi$. We want that, to encounter an accepting location of $\mathcal{B}_\Phi$ in a run, corresponds to see a location of F on all the branches of the corresponding run of $\mathcal{A}_\Phi$. When such an accepting location of $\mathcal{B}_\Phi$ is encountered, we need to start again the marking. The following definition of λ reflects this need. If $\lambda^\triangle \in F^\mathcal{B}$, we let λ be such that: $\forall \ell \in L^\Phi$, $\lambda(\ell) = \{(x, y, \bot) \mid (x, y, \top) \in \lambda^\triangle(\ell)\}$. If $\lambda^\triangle \notin F^\mathcal{B}$, we let $\lambda = \lambda^\triangle$. We

assume the set of 4-tuples (ℓ, x, y, m) such that $(x, y, m) \in \lambda(\ell)$ (for $\ell \in L^\Phi$) is denoted:

$$\{(\ell_1, x_1, y_1, m_1), \ldots, (\ell_k, x_k, y_k, m_k)\}.$$

Remark that several ℓ_i, for $1 \leqslant i \leqslant k$ might be the same location $\ell \in L^\Phi$. Then, $(\lambda^\triangle, \sigma, g, r, \lambda') \in \delta$ iff there is a set $A = \{(a_i)_{i=1}^k\}$ such that:

- For all $i \in \{1, \ldots, k\}$: a_i is an arc of δ^Φ of the form (ℓ, σ, γ_i) associated with $(x_i, y_i, m_i) \in \lambda(\ell_i)$.
- For each $\ell \in L \backslash F$, we let $\lambda^\star(\ell) = \{(x_1^\star, y_1^\star, m_1^\star)(x_2^\star, y_2^\star, m_2^\star) \cdots (x_n^\star, y_n^\star, m_n^\star)\}$ be obtained from $\lambda(\ell)$ by deleting all the trios (x, y, m) such that $(x, y) \notin \bigcup_{i=1}^k \mathsf{loop}\,(a_i)$. For each $\ell \in L \cap F$, we let $\lambda^\star(\ell) = \{(x_1^\star, y_1^\star, \top)(x_2^\star, y_2^\star, \top) \cdots (x_n^\star, y_n^\star, \top)\}$ be obtained from $\lambda(\ell)$ by deleting all the trios (x, y, m) such that $(x, y) \notin \bigcup_{i=1}^k \mathsf{loop}\,(a_i)$. We furthermore suppose that $(x_1^\star, y_1^\star)$ is the last pair of clock that have been associated to location ℓ. Then, for all $\ell \in L$:

  1. *if* $\ell \notin \bigcup_{i=1}^k \mathsf{reset}\,(a_i)$*:*

     $\lambda'(\ell) = \lambda^\star(\ell)$

  2. *else, if* $\ell \in F$ *(in particular, for* $1 \leqslant i \leqslant n$*,* $m_i^\star = \top$*):*

     $\lambda'(\ell) \in \big\{\{(x, y, \top)\} \cup \lambda^\star(\ell)\ ,\ \{(x, y_1^\star, m_1^\star)(x_2^\star, y_2^\star, m_2^\star) \cdots (x_n^\star, y_n^\star, m_n^\star)\}\big\}$

  3. *else, if* $\ell \in \bigcup\limits_{\substack{i \in \{1,\ldots,k\} \\ m_i = \bot}} \mathsf{reset}\,(a_i)$*:*

     $\lambda'(\ell) \in \big\{\{(x, y, \bot)\} \cup \lambda^\star(\ell)\ ,\ \{(x, y_1^\star, \bot)(x_2^\star, y_2^\star, m_2^\star) \cdots (x_n^\star, y_n^\star, m_n^\star)\}\big\}$

  4. *else* (i.e. $\ell \notin \bigcup\limits_{\substack{i \in \{1,\ldots,k\} \\ m_i = \bot}} \mathsf{reset}\,(a_i)$ and $\ell \in \bigcup\limits_{\substack{i \in \{1,\ldots,k\} \\ m_i = \top}} \mathsf{reset}\,(a_i)$)

     $\lambda'(\ell) \in \big\{\{(x, y, \top)\} \cup \lambda^\star(\ell)\ ,\ \{(x, y_1^\star, m_1^\star)(x_2^\star, y_2^\star, m_2^\star) \cdots (x_n^\star, y_n^\star, m_n^\star)\}\big\}$

When $\lambda'(\ell) = \{(x, y, \bot)\} \cup \lambda^\star(\ell)$ or $\{(x, y, \top)\} \cup \lambda^\star(\ell)$, we let $R_\ell = \{x, y\}$; when $\lambda'(\ell) = \{(x, y_1^\star, \bot)(x_2^\star, y_2^\star, m_2^\star) \cdots (x_n^\star, y_n^\star, m_n^\star)\}$ or $\{(x, y_1^\star, m_1^\star)\ (x_2^\star, y_2^\star, m_2^\star) \cdots\ (x_n^\star, y_n^\star, m_n^\star)\}$, we let $R_\ell = \{x\}$; and we let $R_\ell = \varnothing$ otherwise.

- $g = \bigwedge_{1 \leqslant i \leqslant k} (\mathsf{guard}\,(a_i)\,[x/x_i] \wedge \mathsf{guard}\,(a_i)\,[x/y_i])$.
- $r = \bigcup_{\ell \in L} R_\ell$.

The following theorem shows that, for all MITL formula Φ interpreted over infinite words, $\mathcal{B}_\Phi$ is a Büchi timed automaton recognizing $[\![\Phi]\!]$, as we wanted.

Theorem 5.16. *For each MITL formula Φ, $L^\omega(\mathcal{B}_\Phi) = L^\omega_{f^\star_\Phi}(\mathcal{A}_\Phi)$.*

Proof. To prove this, we will show that the transition system $\mathsf{TTS}\,(\mathcal{B}_\Phi) = (\Sigma, S^{\mathsf{TTS}}, s_0^{\mathsf{TTS}}, \rightarrow^{\mathsf{TTS}}, \rightsquigarrow^{\mathsf{TTS}})$ induced by $\mathcal{B}_\Phi$ in the classical semantics (see Definition 2.72) is $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f^\star_\Phi)$ in which $(\lambda, v) \in S^{\mathsf{TTS}}$ corresponds to $\{(\ell, [v(x), v(y)], m) \mid (x, y, m) \in \lambda(\ell)\}$. It is easy to see that the initial configuration of $\mathcal{B}_\Phi$, (λ_0, v_0), where for all $x \in X$, $v_0(x) = 0$, corresponds to the initial state s_0 of $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f^\star_\Phi)$. Now, suppose that we reached a configuration $(\lambda, v) \in S^{\mathsf{TTS}}$ corresponding to the state μ of $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f^\star_\Phi)$, i.e. $\mu = \{(\ell, [v(x), v(y)], m) \mid (x, y, m) \in \lambda(\ell)\}$.

<u>Timed transition:</u> let $t \in \mathbb{R}$: $((\lambda, v), t, (\lambda, v + t)) \in \rightsquigarrow^{\mathsf{TTS}}$ while

$$\mu = \bigcup_{\ell \in L} \{(\ell, [v(x), v(y)], m) \mid (x, y, m) \in \lambda(\ell)\} \overset{t}{\rightsquigarrow}^{\mathsf{MH}} \bigcup_{\ell \in L} \{(\ell, [v(x), v(y)] + t, m) \mid (x, y, m) \in \lambda(\ell)\},$$

the two images correspond.

<u>Discrete transition:</u>

<u>From S^{TTS} to $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f^\star_\Phi)$:</u> Suppose that $((\lambda^\triangle, v), \sigma, (\lambda', v')) \in \rightarrow^{\mathsf{TTS}}$ and that $\mu^\triangle$ corresponds to $(\lambda^\triangle, v)$, i.e. :

$$\mu^\triangle = \bigcup_{\ell \in L} \{(\ell, [v(x), v(y)], m) \mid (x, y, m) \in \lambda^\triangle(\ell)\}.$$

We will show that there exists μ' such that $\mu^\triangle \overset{\sigma}{\rightarrow}^{\mathsf{MH}} \mu'$ in $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f^\star_\Phi)$ and (λ', v') and μ' correspond. As, $((\lambda^\triangle, v), \sigma, (\lambda', v')) \in \rightarrow^{\mathsf{TTS}}$, if $\lambda^\triangle \in F^{\mathcal{B}}$, according

to the previous definition, $\lambda^{\triangle}$ is first turned to

$$\lambda = \{(x, y, \bot) \mid (x, y, \top) \in \lambda(L)\}.$$

In this case, $\mu^{\triangle} \in \alpha$ and it is first turned to

$$\mu = \bigcup_{\ell \in L} \{(\ell, [v(x), v(y)], \bot) \mid (x, y, \top) \in \lambda(\ell)\}$$

(see point (vi) of Definition 5.12) which corresponds to (λ, v). If $\lambda^{\triangle} \notin F^{\mathcal{B}}$, $\mu^{\triangle} \notin \alpha$ and we let $\lambda = \lambda^{\triangle}$ and $\mu = \mu^{\triangle}$.

As $((\lambda^{\triangle}, v), \sigma, (\lambda', v')) \in \rightarrow^{\mathsf{TTS}}$, for all $\ell \in L$, an arc $a_{(x,y)}$ must have been associated with each $(x, y, m) \in \lambda(\ell)$. $\lambda^{\star}$ is then defined in way each (x, y, m) associated with an arc that loop is still associated with the same location while the others disappear. In the last 4 cases (1., 2., 3. and 4.) all the locations to which an arc goes without looping (characterized by the fact that there is a reset through this location in our automaton $\mathcal{A}_{\Phi}$) are considered: either two new clocks are reset and associated with these locations, or one new clock is reset and replaces the clock $x_1^{\star}$ of the first pair of clocks associated with this location (and so, to the smallest represented interval). Whatever is the case thanks to which (λ', v') has been formed, each of the pairs of clocks (x, y) of λ must have satisfied the clock constraints $\mathsf{guard}\left(a_{(x,y)}\right) \wedge \mathsf{guard}\left(a_{(x,y)}\right)[x/y]$, what corresponds to the fact that the whole interval $[v(x), v(y)]$ satisfies $\mathsf{guard}\left(a_{(x,y)}\right)$ (because this guard is an interval and is so convex).

Moreover, v' is obtained from v by associating 0 to each clock reset in the case 1., 2., 3. or 4., used to create (λ', v'), and by associating $v(x)$ to all other clock x. This treatment of (λ, v) to obtain (λ', v') exactly correspond to the fact that

$$\bigcup_{\ell \in L} \{(\ell, [v(x), v(y)]) \mid (x, y, m) \in \lambda(\ell)\} \xrightarrow{\sigma}_{f_{\Phi}^{\star}} \bigcup_{\ell \in L} \{(\ell, [v'(x), v'(y)]) \mid (x, y, m) \in \lambda'(\ell)\}$$

in $\mathcal{A}_{\Phi}$, thanks to the minimal models obtained following, for all (x, y), the arc $a_{(x,y)}$ from $(\ell, [v(x), v(y)])$. We so take:

$$\mu' = \bigcup_{\ell \in L} \{(\ell, [v'(x), v'(y)], m) \mid (x, y, m) \in \lambda'(\ell)\}.$$

It is not difficult to see that, whatever is the case thanks to which (λ', v') has been formed, the marking of pairs of clocks of λ' enables μ' to satisfy conditions

(b) and (c) of the definition of the transition $\rightarrow^{\mathsf{MH}}$ of $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f^\star_\Phi)$. We so have $\mu^\triangle \xrightarrow{\sigma}^{\mathsf{MH}} \mu'$ in $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f^\star_\Phi)$ and (λ', v') and μ' correspond.

From $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f^\star_\Phi)$ to S^{TTS}: Suppose that $\mu^\triangle \xrightarrow{\sigma}^{\mathsf{MH}} \mu'$ in $\mathsf{MHTS}\,(\mathcal{A}_\Phi, f^\star_\Phi)$ and that $(\lambda^\triangle, v)$ and $\mu^\triangle$ correspond, i.e. :

$$\mu^\triangle = \bigcup_{\ell \in L} \{(\ell, [v(x), v(y)], m) \mid (x, y, m) \in \lambda^\triangle(\ell)\}.$$

We will show that there exists (λ', v') such that $((\lambda^\triangle, v), \sigma, (\lambda', v')) \in \rightarrow^{\mathsf{TTS}}$. As $\mu^\triangle \xrightarrow{\sigma}^{\mathsf{MH}} \mu'$, if $\mu^\triangle \in \alpha$, $\mu^\triangle$ is first turned to

$$\mu = \bigcup_{\ell \in L} \{(\ell, [v(x), v(y)], \bot) \mid (x, y, \top) \in \lambda(\ell)\}.$$

In this case, $\lambda^\triangle \in F^{\mathcal{B}}$ and is first turned to

$$\lambda = \{(x, y, \bot) \mid (x, y, \top) \in \lambda(L)\},$$

so that (λ, v) corresponds to μ. If $\mu^\triangle \notin \alpha$, $\lambda^\triangle \notin F^{\mathcal{B}}$ and we let $\mu = \mu^\triangle$ and $\lambda = \lambda^\triangle$.

As $\mu \xrightarrow{\sigma}^{\mathsf{MH}} \mu'$ and μ and (λ, v) correspond,

$$\bigcup_{\ell \in L} \{(\ell, [v(x), v(y)]) \mid (x, y, m) \in \lambda(\ell)\} \xrightarrow{\sigma}_{f^\star_\Phi} \{(\ell, I) \mid (\ell, I, m) \in \mu'\}$$

in $\mathcal{A}_\Phi$ (see Definition 5.12). It means an arc $a_{(x,y)}$ has been chosen for each (x, y) to create a minimal model of $\delta(\ell, \sigma)$ with respect to $[v(x), v(y)]$. We take (λ', v') to be the *unique* successor of (λ, v) in S^{TTS} obtained associating to each (x, y, m) the arc $a_{(x,y)}$ and such that exactly one clock is reset and associated to location ℓ (i.e. R_ℓ is a singleton, in the previous definition of δ, the transition relation of $\mathcal{B}_\Phi$) iff the application of $f^\star_\Phi$ merged the new interval created in location ℓ with the previous smallest one (it is not difficult to see that we indeed obtain a *unique* successor this way, observing the definition of δ, transition relation of $\mathcal{B}_\Phi$, see Definition 5.15).

μ' and (λ', v') correspond, thanks to the fact that the (unique possible) marking present on the third components of elements of μ' will be the same than the obtained marking of elements of (λ', v') (it is easy to see, observing all the cases thanks to which μ', and so (λ', v'), might be constructed). $\square$

The following theorem gives an upper bound on the number of locations of $\mathcal{B}_\Phi$.

Theorem 5.17. *For all MITL formula Φ, $\mathcal{B}_\Phi$ has $M(\Phi)$ clocks and $O((|\Phi|)^{(m.|\Phi|)})$ locations, where $m = \max_{I \in \mathcal{I}_\Phi} \left\{2 \times \left\lceil \frac{\inf(I)}{|I|} \right\rceil + 1, \left\lceil \frac{\sup(I)}{|I|} \right\rceil + 1\right\}$.*

Proof. By definition of $\mathcal{B}_\Phi$, $|X| = M(\Phi)$. Moreover, one location of this automaton is an association, to each location ℓ of $\mathcal{A}_\Phi$, of a finite set $\{(x_1, y_1, m_1), \ldots, (x_n, y_n, m_n)\}$ of trios consisting in a pair of clocks from X and a marker from $\{\top, \bot\}$, such that each pair of clock is associated with a unique location ℓ. In other words, for each couple of clocks (x_i, y_i), either $(x_i, y_i, \top)$ or $(x_i, y_i, \bot)$ is associated with: one and only one $\ell \in L$ or to no $\ell \in L$. For each couple of clocks (x_i, y_i), we so have $2.(|L|+1)$ possibilities of association of $(x_i, y_i, \top)$ or $(x_i, y_i, \bot)$ to a certain, or no, location $\ell \in L$. As there are $\frac{M(\Phi)}{2}$ such pairs of clocks, $\mathcal{B}_\Phi$ has $(2.(|L|+1))^{\frac{M(\Phi)}{2}}$ locations, i.e.: $O\left((|\Phi|)^{m.|\Phi|}\right) = O\left(2^{m.|\Phi|.log_2(|\Phi|)}\right)$ (because $|L| = O(|\Phi|)$ and $M(\Phi) = O(2.m.|\Phi|)$). □

In what precedes, we defined, for all MITL formula Φ, a Büchi automaton $\mathcal{B}_\Phi$ recognizing $[\![\Phi]\!]^\omega$ on the pointwise semantics and discussed its size in Proposition 5.17. We recall that in their paper [5], Alur *and al.* also provide a construction to translate an MITL formula Φ into a timed automaton $\mathcal{B}_\Phi^{cont}$, *over the continuous semantics.* Then, they give an algorithm using a space doubly exponential in the size of Φ to solve the MITL model-checking problem. The sizes of $\mathcal{B}_\Phi$ and $\mathcal{B}_\Phi^{cont}$ are similar, so that our construction of $\mathcal{B}_\Phi$ and Proposition 4.15 formally prove that there is an algorithm using a space doubly exponential in the size of Φ enabling to solve the MITL model-checking problem over the pointwise semantics.

5.3 MITL model-checking with TOCATA: the techniques

From now on, we fix an MITL formula Φ and assume that the OCATA representing the negation of Φ is $\mathcal{A}_{\neg\Phi} = (\Sigma, L, \ell_0, F, \delta)$. We also fix a Büchi TA $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$, and we consider the MITL model-checking problem (see Definition 2.122) and the MITL satisfiability problem (see Definition 2.121) over infinite words. It is of course possible to elaborate such algorithms using the Büchi timed automaton $\mathcal{B}_{\neg\Phi}$, recognizing $[\![\neg\Phi]\!]^{\omega}$ defined is the previous section. Nevertheless, as for the finite words case, we are looking for algorithms working *on the fly*.

Our approach follows the steps of [51], as already discribed and followed in the finite words setting. We first construct a timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi}$ representing the parallel execution of $\mathcal{B}$ and $\mathcal{A}_{\neg\Phi}$. It is obtained by first taking the synchronous product of $\mathsf{TTS}\,(\mathcal{A}_{\neg\Phi}, f^{\star}_{\neg\Phi})$ and the transition system $\mathsf{TTS}\,(\mathcal{B})$ of $\mathcal{B}$ (see Definition 2.72), and then associating Miyano-Hayashi markers with its states, by adapting the construction of $\mathsf{MHTS}\,(\mathcal{A}, f)$ to cope with the configurations of $\mathcal{B}$. Then, the aim will be to verify that $\mathcal{S}_{\mathcal{B},\neg\Phi}$ has no accepting run, i.e. no *reachable* accepting state of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is *reachable from itself*. We can symmetrically solve the *MITL satisfiability problem* by looking for accepting run in the timed transition system $\mathcal{S}_{\mathcal{A}_{\neg\Phi}, f^{\star}_{\neg\Phi}}$ (see Definition 3.16). Since the techniques are similar for model-checking and satisfiability (see Section 2.4.2), we will only detail the former in this section. As for finite words, the timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is infinitely branching and we must use a region abstraction. The one presented here only changes from that exhibited over finite words by the addition of Miyano-Hayashi markers. Each region will still be symbolically represented by a unique word.

For the sake of simplicity, we keep the same name for the present timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi}$ than for that defined in Section 4.2, over finite words.

Let us formally define $\mathcal{S}_{\mathcal{B},\neg\Phi}$, representing the parallel execution of $\mathcal{B}$ and $\mathcal{A}_{\neg\Phi}$.

Definition 5.18. *Let Φ be an MITL formula and $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$ be a Büchi timed automaton. We define the timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi} = (\Sigma, S, s_0, \rightsquigarrow, \rightarrow, \alpha)$ where:*

(i) S is the set of elements of the form $\{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_{\mathcal{B}})\}$ where $\{(\ell_k, I_k)_{k\in K}\}$ is a configuration of $\mathcal{A}_{\neg\Phi}$, (b, v) is a state of $\mathcal{B}$, $m_{\mathcal{B}} \in \{\top, \bot\}$ and $\forall k \in K, m_k \in \{\top, \bot\}$;

(ii) $s_0 = \{(\ell_0, [0, 0], \bot), (b_0, v_0, m)\}$, where v_0 is the valuation such that $v_0(x) = 0$, $\forall x \in X$, and $m = \top$ iff $b_0 \in F^{\mathcal{B}}$;

(iii) α contains all the elements of S of the form $\{(\ell_k, I_k, \top)_{k\in K}\} \cup \{(b, v, \top)\}$;

(iv) the transition relation $\rightsquigarrow$ takes care of the elapsing of time: $\forall t \in \mathbb{R}$ and $s, s' \in S$, supposing $s = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_{\mathcal{B}})\}$, we have $s \overset{t}{\rightsquigarrow} s'$ iff $s' = \{(\ell_k, I_k + t, m_k)_{k\in K}\} \cup \{(b, v + t, m_{\mathcal{B}})\}$. $\rightsquigarrow = \bigcup_{t\in\mathbb{R}} \overset{t}{\rightsquigarrow}$;

(v) for $s \in S\backslash\alpha$ and $s' \in S$, supposing $s = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_{\mathcal{B}})\}$ and $s' = \{(\ell_{k'}, I_{k'}, m_{k'})_{k'\in K'}\} \cup \{(b', v', m'_{\mathcal{B}})\}$, $s \xrightarrow{\sigma} s'$ iff

(a) $\{(\ell_k, I_k)_{k\in K}\} \xrightarrow{\sigma}_{f^\star_{\neg\Phi}} \{(\ell_{k'}, I_{k'})_{k'\in K'}\}$ in $\mathcal{A}_{\neg\Phi}$, i.e. $\{(\ell_{k'}, I_{k'})_{k'\in K'}\} \in f^\star_{\neg\Phi}(\mathsf{Succ}(\{(\ell_k, I_k)_{k\in K}\}, \sigma))$, and $(b, v) \xrightarrow{\sigma} (b', v')$ in $\mathcal{B}$;

(b) $\forall k' \in K'$: $(\ell_{k'} \in F \Rightarrow m_{k'} = \top)$;

(c) $\forall \overline{k} \in K'$ with $\ell_{\overline{k}} \notin F$: if $\exists k^* \in K$ such that $(\ell_{k^*}, I_{k^*}, \bot) \in s$ and $(\ell_{\overline{k}}, I_{\overline{k}}) \in \mathsf{dest}(\{(\ell_k, I_k)_{k\in K}\}, \{(\ell_{k'}, I_{k'})_{k'\in K'}\}, (\ell_{k^*}, I_{k^*}))$, we have $m_{\overline{k}} = \bot$; otherwise, $m_{\overline{k}} = \top$;

(d) $m'_{\mathcal{B}} = \top$ iff $m_{\mathcal{B}} = \top$ or $b' \in F^{\mathcal{B}}$.

(vi) For $s \in \alpha$ and $s' \in S$, supposing $s = \{(\ell_k, I_k, \top)_{k\in K}\} \cup \{(b, v, \top)\}$, $s \xrightarrow{\sigma} s'$ iff $\{(\ell_k, I_k, \bot)_{k\in K}\} \cup \{(b, v, \bot)\} \xrightarrow{\sigma} s'$ according to the rules in (i) ;
$\rightarrow = \bigcup_{\sigma\in\Sigma} \xrightarrow{\sigma}$.

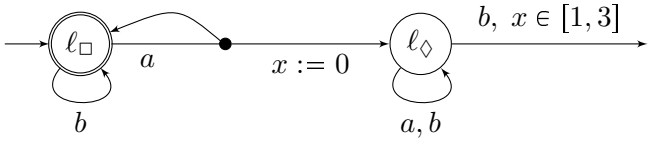


Figure 5.3: OCATA $\mathcal{A}_{\neg\Phi}$ with $\neg\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,3]} b)$.

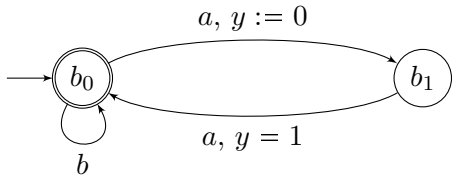


Figure 5.4: A timed automaton $\mathcal{B}$.

The transition relation '$\rightarrow$' of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is defined in way that (1) a discrete transition between two states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ corresponds to a discrete transition between the configurations of $\mathcal{A}_{\neg\Phi}$ they contain, (2) a discrete transition between the states of $\mathcal{B}$ they contain and (3) the markers of the third component are kept updated.

Example 5.19. We consider the Büchi timed automaton $\mathcal{B}$ of Figure 5.4 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 5.3, for the MITL formula $\neg\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,3]} b)$. Figure 5.5 gives a part of the timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi}$. Its construction is similar to that given over finite words (see Example 4.18 for details), but the markers must be kept updated. For instance, as the overhead state is accepting, the marking is started again reading a or b. As $\ell_\Box$ is the unique accepting location in the state reached from it reading an a, its trio is the unique marked by $\top$.

The following proposition ensures $\mathcal{S}_{\mathcal{B},\neg\Phi}$ correctly represents the parallel execution of $\mathcal{B}$ and $\mathcal{A}_{\neg\Phi}$. The proof is similar to that of Proposition 5.13 and can be found in the appendix.

Proposition 5.20. For every MITL formula Φ, the associated $\mathcal{A}_{\neg\Phi}$ and $f^{\star}_{\neg\Phi}$,

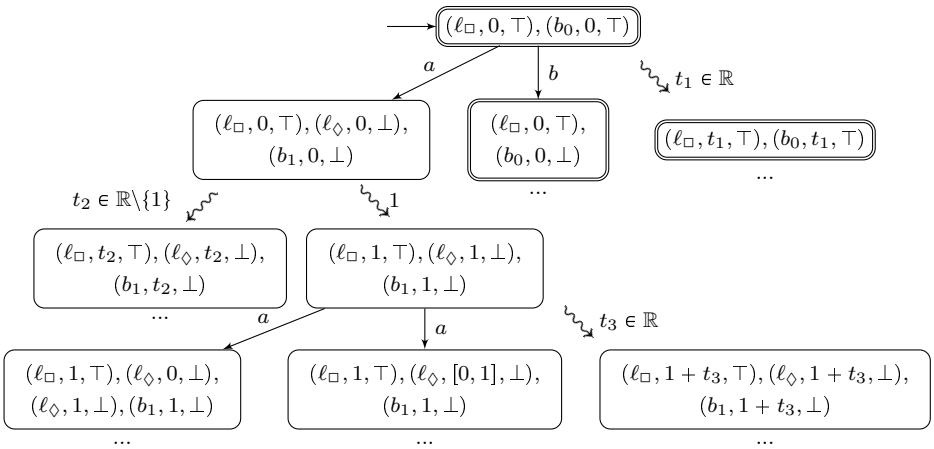


Figure 5.5: Representation of a part of $\mathcal{S}_{\mathcal{B},\neg\Phi}$.

and for every Büchi timed automaton $\mathcal{B}$:

$$L^\omega(\mathcal{S}_{\mathcal{B},\neg\Phi}) = L^\omega_{f^\star_{\neg\Phi}}(\mathcal{A}_{\neg\Phi}) \cap L(\mathcal{B}).$$

5.4 Region-based algorithm

The aim of this section is to provide a region-based algorithm to solve the MITL model-checking problem (see Definition 2.122). Thanks to Proposition 5.20, such an algorithm only need to check that $\mathcal{S}_{\mathcal{B},\neg\Phi}$ has no accepting run. Nevertheless, the timed transition system $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is infinitely branching and we must use a region abstraction: we adapt the region abstraction presented on finite words in order to cope with the Miyano-Hayashi markers. Once again, the aim is to symbolically represent each region by a unique word.
For the sake of simplicity, we keep the same names for the region equivalence $\equiv$, and the function $\mathcal{H}$ giving a symbolic representation of each of its classes, than for those defined in Section 4.2, over finite words.

Here is the definition of the equivalence relation $\equiv$ on S. It is similar to

Definition 4.21: once again the states of the two configurations of $\mathcal{A}_{\neg\Phi}$ are indexed on a same set K in aim to make 'correspond' the states of same index. The only difference with Definition 4.21 is the addition, in point 1., of a condition that forces corresponding trios of the two states of S to hold the same markers.

Definition 5.21. *Let Φ be an MITL formula, $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$ be a timed automaton and $\mathcal{S}_{\mathcal{B},\neg\Phi}$, of state space S, be the transition system given by Definition 5.18. Let K be a set of indices and let s and s' be two states of S such that the configurations of $\mathcal{A}_{\neg\Phi}$ they contain have the same cardinality. We suppose that $s = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_{\mathcal{B}})\}$ and $s' = \{(\ell'_k, I'_k, m'_k)_{k\in K}\} \cup \{(b', v', m'_{\mathcal{B}})\}$. Then, we define $s \equiv s'$ iff:*

1. $b = b'$ *and* $\forall k \in K : \ell_k = \ell'_k$; $m_{\mathcal{B}} = m'_{\mathcal{B}}$ *and* $\forall k \in K : m_k = m'_k$,
2. $\forall 1 \leqslant p \leqslant n : v(x_p) \sim v'(x_p)$ *and* $\forall k \in K : (\inf(I_k) \sim \inf(I'_k) \wedge \sup(I_k) \sim \sup(I'_k))$,
3. $\forall 1 \leqslant p, q \leqslant n : frac(v(x_p)) \bowtie frac(v(x_q))$ *iff* $frac(v'(x_p)) \bowtie frac(v'(x_q))$,
4. $\forall k, k' \in K : frac(\inf(I_k)) \bowtie frac(\inf(I_{k'}))$ *iff* $frac(\inf(I'_{k'})) \bowtie frac(\inf(I'_{k'}))$,
5. $\forall k, k' \in K : frac(\sup(I_k)) \bowtie frac(\sup(I_{k'}))$ *iff* $frac(\sup(I'_k)) \bowtie frac(\sup(I'_{k'}))$,
6. $\forall k, k' \in K : frac(\inf(I_k)) \bowtie frac(\sup(I_{k'}))$ *iff* $frac(\inf(I'_k)) \bowtie frac(\sup(I'_{k'}))$,
7. $\forall k \in K, \forall 1 \leqslant p \leqslant n : frac(\inf(I_k)) \bowtie frac(v(x_p))$ *iff* $frac(\inf(I'_k)) \bowtie frac(v'(x_p))$,
8. $\forall k \in K, \forall 1 \leqslant p \leqslant n : frac(\sup(I_k)) \bowtie frac(v(x_p))$ *iff* $frac(\sup(I'_k)) \bowtie frac(v'(x_p))$,

where $\bowtie \in \{<, =, >\}$.

Example 5.22. We consider again the TA $\mathcal{B}$ of Figure 5.4 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 5.3, for the MITL formula $\neg\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,3]} b)$. We consider the

following states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$:

$$\begin{aligned} s_1 &:= \{(\ell_\Box, 1.4, \top), (\ell_\Diamond, [0, 0.2], \bot), (b_1, 0.7, \bot)\} \\ s_2 &:= \{(\ell_\Box, 1.6, \top), (\ell_\Diamond, [0, 0.5], \bot), (b_1, 0.65, \bot)\} \\ s_3 &:= \{(\ell_\Box, 1.6, \top), (\ell_\Diamond, [0, 0.5], \bot), (b_1, 0.65, \top)\}. \end{aligned}$$

On the one hand, we have $s_1 \equiv s_2$. Indeed, we make correspond $(\ell_\Box, 1.4, \top)$ with $(\ell_\Box, 1.6, \top)$, $(\ell_\Diamond, [0, 0.2], \bot)$ with $(\ell_\Diamond, [0, 0.5], \bot)$ and $(b_1, 0.7, \bot)$ with $(b_1, 0.65, \bot)$ in way condition 1. is satisfied. Condition 2. is satisfied because $1.4 \sim 1.6$ (class $]1, 2[$), $0 \sim 0$ (class $\{0\}$), $0.2 \sim 0.5$ (class $]0, 1[$) and $0.7 \sim 0.65$ (class $]0, 1[$). Sorting the clock values of s_1 following the increasing order of their fractional parts, we obtain:

$$frac(0) < frac(0.2) < frac(1.4) < frac(0.7).$$

In s_2, we obtain:

$$frac(0) < frac(0.5) < frac(1.6) < frac(0.65).$$

As corresponding clocks values are in the same place in these orderings, one can verify that conditions 3. to 8. are indeed satisfied. For instance, condition 4. holds because, for each $\bowtie \in \{<, =, >\}$:

$$\begin{aligned} & frac(1.4) = 0.4 \bowtie frac(0) = 0 \quad \text{iff} \quad frac(1.6) = 0.6 \bowtie frac(0) = 0, \\ & frac(1.4) = 0.4 \bowtie frac(0.7) = 0.7 \quad \text{iff} \quad frac(1.6) = 0.6 \bowtie frac(0.65) = 0.65 \\ \text{and} \quad & frac(0) = 0 \bowtie frac(0.7) = 0.7 \quad \text{iff} \quad frac(0) = 0 \bowtie frac(0.65) = 0.65. \end{aligned}$$

On the second hand, $s_1 \not\equiv s_3$. Indeed, we must make correspond $(\ell_\Box, 1.4, \top)$ with $(\ell_\Box, 1.6, \top)$, $(\ell_\Diamond, [0, 0.2], \bot)$ with $(\ell_\Diamond, [0, 0.5], \bot)$ and $(b_1, 0.7, \bot)$ with $(b_1, 0.7, \top)$ to satisfy the beginning of condition 1.. Nevertheless, this does not enable to entirely satisfy condition 1. because the markers of $(b_1, 0.7, \bot)$ and $(b_1, 0.7, \top)$ do not match.

The equivalence relation $\equiv$ induces a time-abstract bisimulation on the states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$, as stated by the following proposition. Its proof is close to that of Proposition 4.23, the unique supplementary difficulty is to handle the Miyano-Hayashi markers: it can be found in the appendix.

Proposition 5.23 (Time-abstract bisimulation)**.** Let Φ be an MITL formula, $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$ be a timed automaton and $\mathcal{S}_{\mathcal{B},\neg\Phi}$, of state space S, be the transition system given by Definition 5.18. Let $s_1, s_2 \in S$ such that $s_1 \equiv s_2$. Then:

1. for each transition
   $s_1 \overset{t}{\rightsquigarrow} z_1$ with $t \in \mathbb{R}^+$ and $z_1 \in S$,
   there exists $t' \in \mathbb{R}^+$ and $z_2 \in S$ such that:
   $s_2 \overset{t'}{\rightsquigarrow} z_2 \quad$ and $\quad z_1 \equiv z_2$;

2. for each transition
   $s_1 \xrightarrow{\sigma} z_1$, with $\sigma \in \Sigma$ and $z_1 \in S$,
   there exists $z_2 \in S$ such that:
   $z_1 \xrightarrow{\sigma} z_2 \quad$ and $\quad z_1 \equiv z_2$.

As in the finite words setting, we remark that the size of the configurations of $\mathcal{A}_{\neg\Phi}$ we can encounter in $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is bounded by $M(\neg\Phi)$, thanks to the use of $f^{\star}_{\neg\Phi}$. So, there is only a finite number of such configurations. As a consequence, the number of configurations of $\mathcal{A}_{\neg\Phi}$ we can encounter in $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is finite. As the number of regions is also finite, the quotient of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ by $\equiv$ is finite and we can elaborate a model-checking algorithm using it.
In the sequel, we define a symbolic representation of each of these regions by a unique word, in a similar way as for finite words. Nevertheless, the definition of these words must be adapted: an additional component must be added to represent the Miyano-Hayashi markers. Before giving the formal definition, let us explain how we proceed on an example.

Example 5.24. Let us consider the TA $\mathcal{B}$ of Figure 5.4 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 5.3. To make this example interesting, we consider the following unreachable state of $\mathcal{S}_{\mathcal{B},\neg\Phi}$:

$$s_1 \ := \ \{(b_1, 1.5, \top), (\ell_\Diamond, [0, 0.5], \bot), (\ell_\Diamond, [1.7, 3.3], \bot)\}.$$

To represent the region of s_1 according to $\equiv$, we are interested in each of the values of clocks and clock copies 1.5, 0, 0.5, 1.7 and 3.3, as well as in the order of their fractional parts. Furthermore, it is important to remember that state $(b_1, 1.5)$ is marked by $\top$, and that values 0 and 0.5 form an interval marked by $\bot$, as well as values 1.7 and 3.3.

We first construct a 4-tuple to represent each value of the clock copies of $\mathcal{A}_{\neg\Phi}$. A 4-tuple contains the location to which the clock copy is associated, the value of the clock copy, the associated Miyano-Hayashi marker and a number to remember which couples of clock copies form intervals. For s_1, we have:

$$\begin{array}{llll} \text{value} & 0 & \text{is represented by} & (\ell_\Diamond, 0, \bot, 1), \\ \text{value} & 0.5 & \text{is represented by} & (\ell_\Diamond, 0.5, \bot, 1), \\ \text{value} & 1.7 & \text{is represented by} & (\ell_\Diamond, 1.7, \bot, 2), \\ \text{value} & 3.3 & \text{is represented by} & (\ell_\Diamond, 3.3, \bot, 2). \end{array}$$

The value of clocks of $\mathcal{B}$ are also represented by a 4-tuple containing the location in which $\mathcal{B}$ is, the value of the nth clock of $\mathcal{B}$, a marker n to remember to which clock of $\mathcal{B}$ the value corresponds, and the associated Miyano-Hayashi marker. For s_1, we have:

$$\text{value} \quad 1.5 \quad \text{is represented by} \quad (b_1, 1.5, \top, 1).$$

Secondly, we sort all the obtained 4-tuples in different sets: we create a set for each value of the fractional parts of the clock values. We recall that the fractional part of clock (copies) values beyond c_{max} (= 3, here) is considered to be 0. For our example, we have:

$$\begin{array}{l} \{(b_1, 1.5, \top, 1), (\ell_\Diamond, 0.5, \bot, 1)\} \\ \{(\ell_\Diamond, 0, \bot, 1), (\ell_\Diamond, 3.3, \bot, 2)\} \\ \{(\ell_\Diamond, 1.7, \bot, 2)\} \end{array}$$

When the values of clocks and clock copies will be replaced by the region of REG they are in, each of these sets will be a *letter* of the word symbolically representing the region of s_1 according to $\equiv$. Nevertheless, before replacing the values by the good regions of REG, we must sort the letters of this word. In

fact, we sort these letters according to the order of the fractional part of clock values they represent. We here obtain:

$$\{(\ell_\diamond, 0, \bot, 1), (\ell_\diamond, 3.3, \bot, 2)\} \quad \{(b_1, 1.5, \top, 1), (\ell_\diamond, 0.5, \bot, 1)\} \quad \{(\ell_\diamond, 1.7, \bot, 2)\}$$

The region of s_1 is so symbolically represented by:

$$\{(\ell_\diamond, \{0\}, \bot, 1), (\ell_\diamond,]3, +\infty[, \bot, 2)\} \quad \{(b_1,]1, 2[, \top, 1), (\ell_\diamond,]0, 1[, \bot, 1)\}$$
$$\{(\ell_\diamond,]1, 2[, \bot, 2)\}$$

Formally, we encode regions of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ by finite words whose letters are finite sets of 4-tuples of the form (ℓ, r, m, k), where $\ell \in L \cup L^{\mathcal{B}}$, $r \in REG$, $m \in \{\top, \bot\}$ and $0 \leqslant k \leqslant M(\neg\Phi)/2$. Here is the definition of the function H that associates with each $s \in S$ the region it is in: it is similar to the definition given in the setting of finite words.

Definition 5.25. *For $s = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m)\}$, $H(s) = H_1 H_2 \cdots H_p$ is defined as follows:*

1. *For each location ℓ, let $C(\ell) = \{(\ell', I, m) \in C \mid \ell' = \ell\}$. Assume $C(\ell) = \{(\ell_1, I_1, m_1), \ldots, (\ell_k, I_k, m_k)\}$, with $I_1 \leqslant \cdots \leqslant I_k$. Then, we first build $E_\ell = \{(\ell_i, \inf(I_i), m_i, i), (\ell_i, \sup(I_i), m_i, i) \mid 1 \leqslant i \leqslant k\}$.*

2. *We treat $(\ell^{\mathcal{B}}, v, m)$ symmetrically, and let $E^{\mathcal{B}} = \{(\ell^{\mathcal{B}}, v(x_1), m, 1), \ldots, (\ell^{\mathcal{B}}, v(x_n), m, n)\}$. We let $\mathcal{E} = E^{\mathcal{B}} \cup_{\ell\in L} E_\ell$. That is, all elements in $\mathcal{E}$ are tuples (ℓ, v, m, i), where ℓ is a location (of $\mathcal{A}_\Phi$ or $\mathcal{B}$), v is a real value (interval endpoint or clock value), m is a Miyano-Hayashi marker and i is bookkeeping information that links v to an interval (if ℓ is a location of $\mathcal{A}_\Phi$), or to a clock (ℓ is a location of $\mathcal{B}$).*

3. *We partition $\mathcal{E}$ into $\mathcal{E}_1, \ldots, \mathcal{E}_p$ such that each $\mathcal{E}_i$ contains all elements from $\mathcal{E}$ with the same fractional part of their second component (recall that we assume $frac(u) = 0$ for all $u > c_{max}$). We assume the ordering $\mathcal{E}_1$, $\mathcal{E}_2$,..., $\mathcal{E}_p$ reflects the increasing ordering of the fractional parts.*

4. *For all $1 \leqslant i \leqslant p$, we obtain H_i from $\mathcal{E}_i$ by replacing the second component of all elements in $\mathcal{E}_i$ by the region from REG they belong to.*

Thus, noting $\max_\ell$ the maximal number of interval that can be present in location $\ell \in L$ of $\mathcal{A}_{\neg\Phi}$ (given by the proof of Theorem 5.11), and noting $n = |X|$ the number of clocks of $\mathcal{B}$, $H(s)$ will be a finite word over the alphabet $\Lambda = (B \cup L) \times REG \times \{\top, \bot\} \times \{1, 2, \ldots, \max(\max_{\ell \in L}(\max_\ell), n)\}$. We will also view H as a function $H : S \to \Lambda^\star$.

Example 5.26. Consider a TA $\mathcal{B}$ with 1 clock, let $c_{\max} = 2$, and let $s = \big\{\{(\ell_1, [0, 1.3], \bot), (\ell_1, [1.8, 2.7], \top)\}, (\ell^{\mathcal{B}}, 0.3, \bot)\big\}$. The first step of the construction yields the set $\mathcal{E} = \{\, (\ell_1, 0, \bot, 1), (\ell_1, 1.3, \bot, 1), (\ell_1, 1.8, \top, 2), (\ell_1, 2.7, \top, 2), (\ell^{\mathcal{B}}, 0.3, \bot, 1)\,\}$. Then, we have $H(s) = \big\{\, (\ell_1, \{0\}, \bot, 1), (\ell_1,]2, +\infty[, \top, 2)\,\big\}$ $\big\{(\ell_1,]1, 2[, \bot, 1), (\ell^{\mathcal{B}},]0, 1[, \bot, 1)\big\}\big\{(\ell_1,]1, 2[, \top, 2)\big\}$.

As we follows the same reasoning as for finite words, the following definitions and propositions have their twin in the previous chapter. The proofs of the propositions are omitted here because they are similar to the proofs of the corresponding ones for the finite words setting: they can be found in the appendix.

We first present a proposition stating that the words obtained from H give a correct representation of classes of $\equiv$.

Proposition 5.27. Let $s, s' \in S$. We have: $s \equiv s'$ iff $H(s) = H(s')$.

The bisimulation lemma enables to only consider the classes of $\equiv$ (i.e. the words given by function H) instead of all the states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$. We here define the set of classes of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ we are interested in and the (timed and discrete) transitions between those classes, in a similar way as for finite words.

Definition 5.28. *We define:*

$$\mathcal{H} \;=\; \mathcal{S}_{\mathcal{B},\neg\Phi}/\equiv \;=\; \{H(s) \mid s \in S\}.$$

For all $W^1, W^2 \in \mathcal{H}$ *and* $\sigma \in \Sigma$ *we define* $W^1 \xrightarrow{\sigma} W^2$ *iff* $\forall s^1 \in (H)^{-1}(W^1)$, $\exists s^2 \in (H)^{-1}(W^2)$: $s^1 \xrightarrow{\sigma} s^2$.
For all $W^1, W^2 \in \mathcal{H}$, *we define* $W^1 \longrightarrow_T W^2$ *iff* $\forall s^1 \in (H)^{-1}(W^1)$, $\exists t \in \mathbb{R}$ *and* $\exists s^2 \in (H)^{-1}(W^2)$: $s^1 \overset{t}{\rightsquigarrow} s^2$.

Example 5.29. Once again, we consider the TA $\mathcal{B}$ of Figure 5.4 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 5.3. Let us consider the state $\{(\ell_\Box, 2.2, \top), (\ell_\Diamond, [1,2], \bot), (b_0, 2, \top)\}$ of $\mathcal{S}_{\mathcal{B},\neg\Phi}$. Its class in $\mathcal{H}$ is the word:

$$W^1 := \{(\ell_\Diamond, \{1\}, \bot, 1), (\ell_\Diamond, \{2\}, \bot, 1), (b_0, \{2\}, \top, 1)\}\ \{(\ell_\Box,]2,3[, \top, 1)\}.$$

Let us note:

$$W^2 := \{(b_0, \{2\}, \top, 1)\}\ \{(\ell_\Box,]2,3[, \top, 1)\}.$$

We have $W^1 \xrightarrow{b} W^2$. Indeed, let $s^1 \in H^{-1}(W^1)$: s^1 will be of the form $\{(\ell_\Box, t, \top), (\ell_\Diamond, [1,2], \bot), (b_0, 2, \top)\}$, for a certain $t \in]2,3[$. Let us consider $s^2 = \{(\ell_\Box, t, \top), (b_0, 2, \top)\}$, we have that $s^2 \in H^{-1}(W^2)$ and $s^1 \xrightarrow{b} s^2$.
Now, let us note:

$$W^3 := \{(\ell_\Diamond,]1,2[, \bot, 1), (\ell_\Diamond,]2,3[, \bot, 1), (b_0,]2,3[, \top, 1)\}\ \{(\ell_\Box,]2,3[, \top, 1)\}.$$

We have $W^1 \longrightarrow_T W^3$. Indeed, let $s^1 \in H^{-1}(W^1)$: s^1 will be of the form $\{(\ell_\Box, t, \top), (\ell_\Diamond, [1,2], \bot), (b_0, 2, \top)\}$, for a certain $t \in]2,3[$. Let us consider $t' \in]0,1[$ and $s^3 = \{(\ell_\Box, t+t', \top), (\ell_\Diamond, [1+t', 2+t'], \bot), (b_0, 2+t', \top)\}$ such that $t+t' \in]2,3[$ (such a t' always exists as we are using open intervals of $\mathbb{R}$). Hence, we have $s^3 \in H^{-1}(W^3)$ and $s^1 \overset{t'}{\rightsquigarrow} s^3$.

We defined that $W^1 \xrightarrow{\sigma} W^2$ iff **for all** $s^1 \in H^{-1}(W^1)$:

$$\exists s^2 \in H^{-1}(W^2) \text{ such that } s^1 \xrightarrow{\sigma} s^2. \tag{5.1}$$

Indeed, no matter the choice of $s^1 \in H^{-1}(W^1)$, the following proposition states that, if property 5.1 is true for *one* such s^1, it will consequently be true for *all* such s^1.

Proposition 5.30. Let $W^1, W^2 \in \mathcal{H}$, $\sigma \in \Sigma$ and $t \in \mathbb{R}^+$.
$W^1 \xrightarrow{\sigma} W^2$ iff $\exists s^1 \in (H)^{-1}(W^1)$ and $s^2 \in (H)^{-1}(W^2) : s^1 \xrightarrow{\sigma} s^2$.

As previously explained, our model-checking algorithm will consist in looking for a path to an *accepting reachable* state of $\mathcal{H}$ which is *reachable from itself.* We will conclude that $\mathcal{B} \not\models \Phi$ if and only if such a path exists. We so define $Post(\mathcal{W})$, for $\mathcal{W} \subseteq \mathcal{H}$: this operator will be used in our algorithm to compute the successors of the set of states of $\mathcal{H}$ we have already reached.

Definition 5.31. *Let $\mathcal{W} \subseteq \mathcal{H}$, we define:*

$$Post(\mathcal{W}) := \{W' \in \mathcal{H} \mid \exists \sigma \in \Sigma,\ W \in \mathcal{W} \text{ and } W'' \in \mathcal{H} : W \longrightarrow_T W'' \xrightarrow{\sigma} W'\}.$$

To ensure the termination of our following algorithm, we need $Post(W)$ to be finite and effectively computable, for any word $W \in \mathcal{H}$. This is what claims the following proposition. Its proof, presented in the appendix, is similar to that of Proposition 4.31. The procedure to compute $Post(W)$ from W is also similar to that depicted in the proof of this Proposition 4.31.

Proposition 5.32. For each word $W \in \mathcal{H}$, $Post(W)$ is finite and effectively computable.

Example 5.33. We consider again the TA $\mathcal{B}$ of Figure 5.4 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 5.3. Let us consider the following word of $\mathcal{H} = \mathcal{S}_{\mathcal{B},\neg\Phi}/\equiv$:

$$W := \{(\ell_\Box,]3, +\infty[, \top, 1), (\ell_\Diamond, \{2\}, \bot, 1), (\ell_\Diamond, \{3\}, \bot, 1), (b_0, \{3\}, \top, 1)\}.$$

We will look for $Post(W)$. The procedure of the proof of Proposition 4.31 recursively gives the following timed successors of W:

$$\begin{array}{lcl}
W & := & \{(\ell_\Box,]3, +\infty[, \top, 1), (\ell_\Diamond, \{2\}, \bot, 1), (\ell_\Diamond, \{3\}, \bot, 1), (b_0, \{3\}, \top, 1)\}, \\
W^1 & := & \{(\ell_\Box,]3, +\infty[, \top, 1), (\ell_\Diamond,]3, +\infty[, \bot, 1), (b_0,]3, +\infty[, \top, 1)\}\{(\ell_\Diamond,]2, 3[, \bot, 1)\}, \\
W^2 & := & \{(\ell_\Box,]3, +\infty[, \top, 1), (\ell_\Diamond,]3, +\infty[, \bot, 1), (b_0,]3, +\infty[, \top, 1), (\ell_\Diamond, \{3\}, \bot, 1)\}, \\
W^3 & := & \{(\ell_\Box,]3, +\infty[, \top, 1), (\ell_\Diamond,]3, +\infty[, \bot, 1), (b_0,]3, +\infty[, \top, 1), (\ell_\Diamond,]3, +\infty[, \bot, 1)\}.
\end{array}$$

In way to compute the discrete successors of W, we might consider

$$s = \{(\ell_\Box, 3.1, \top), (\ell_\Diamond, [2, 3], \bot), (b_0, 3, \top)\},$$

as $H(s) = W$. Let us note:

$$\begin{array}{rcl} s^1 & := & \{(\ell_\Box, 3.1, \top), (\ell_\Diamond, 0, \bot), (\ell_\Diamond, [2,3], \bot), (b_1, 0, \bot)\}, \\ s^2 & := & \{(\ell_\Box, 3.1, \top), (\ell_\Diamond, [0,3], \bot), (b_1, 0, \bot)\}, \\ s^3 & := & \{(\ell_\Box, 3.1, \top), (\ell_\Diamond, [2,3], \bot), (b_0, 3, \top)\}, \\ s^4 & := & \{(\ell_\Box, 3.1, \top), (b_0, 3, \top)\}. \end{array}$$

We have that $s \xrightarrow{a} s^1$, $s \xrightarrow{a} s^2$, $s \xrightarrow{b} s^3$ and $s \xrightarrow{b} s^4$. This gives the following elements of $Post(W)$:

$$\begin{array}{rcl} W^1_{post} & := & \{(\ell_\Box,]3, +\infty[, \top, 1), (\ell_\Diamond, \{0\}, \bot, 1), (\ell_\Diamond, \{2\}, \bot, 2), (\ell_\Diamond, \{3\}, \bot, 2), (b_1, \{0\}, \bot, 1)\}, \\ W^2_{post} & := & \{(\ell_\Box,]3, +\infty[, \top, 1), (\ell_\Diamond, \{0\}, \bot, 1), (\ell_\Diamond, \{3\}, \bot, 1), (b_1, \{0\}, \bot, 1)\}, \\ W^3_{post} & := & \{(\ell_\Box,]3, +\infty[, \top, 1), (\ell_\Diamond, \{2\}, \bot, 1), (\ell_\Diamond, \{3\}, \bot, 1), (b_0, \{3\}, \top, 1)\}, \\ W^4_{post} & := & \{(\ell_\Box,]3, +\infty[, \top, 1), (b_0, \{3\}, \top, 1)\}. \end{array}$$

A similar procedure enables to compute the others elements of $Post(W)$ from W^1, W^2 and W^3.

In our algorithm, we will be looking for a *reachable* state of $\mathcal{H}$ *reachable from itself*. To algorithmically verify this, we need the following operators $Post^*$ and $Post^+$.

Definition 5.34. *Let $\mathcal{W} \subseteq \mathcal{H}$ and $n \in \mathbb{N}$, we define: $Post^n(\mathcal{W}) = Post(Post^{n-1}(\mathcal{W}))$, with $Post^0(\mathcal{W}) = \mathcal{W}$ and $Post^1(\mathcal{W}) = Post(\mathcal{W})$.*
We define $Post^\star(\mathcal{W}) = \bigcup_{n\in\mathbb{N}} Post^n(\mathcal{W})$ and $Post^+(\mathcal{W}) = \bigcup_{n\in\mathbb{N}_0} Post^n(\mathcal{W})$.

Remark that, as $\mathcal{H}$ is finite, $\exists m \in \mathbb{N} : Post^\star(\mathcal{W}) = \bigcup_{n=1}^m Post^n(\mathcal{W})$.

We still fix some notations and vocabulary used in the presentation of our algorithm.

Definition 5.35. *We note $H_0 := H(s_0)$ the word of $\mathcal{H}$ corresponding to the initial state of $\mathcal{S}_{\mathcal{B},\neg\Phi}$.*

Definition 5.36. *We say that a word $W \in \mathcal{H}$ is* accepting *iff the third components of all the 4-tuples it contains are $\top$ (such words correspond to accepting states of*

$\mathcal{S}_{\mathcal{B},\neg\Phi}$).
We note $\mathcal{F} \subseteq \mathcal{H}$ the set of accepting words of $\mathcal{H}$.

Algorithm 10 is an (classical) algorithm for the model-checking of MITL in which the reachable words of $\mathcal{H} = \mathcal{S}_{\mathcal{B},\neg\Phi}/\equiv$ *reachable from themselves* are computed 'on the fly'.

Algorithm 10 MITLModelCheckingOverInfiniteWords

Input: A TA $\mathcal{B}$ and the ATA $\mathcal{A}_{\neg\Phi}$, for $\Phi \in$ MITL.
Output: 'true' iff $\mathcal{B} \models \Phi$.

```
C ← ∅
D ← Post*(H_0) ∩ F
while C ≠ D do
    C ← D
    D ← Post+(D) ∩ F
end while
if D = ∅ then
    return true
else
    return false
end if
```

We use the following theorem to prove that this algorithm is correct.

Theorem 5.37 ([45] - Theorem 2.3.20)**.** *Let $\mathcal{B} = (\Sigma, L, \ell_0, X, F, \delta)$ be a non-deterministic Büchi automaton.*

$$L(\mathcal{B}) = \emptyset \quad \textit{iff} \quad GFP(\lambda X.Post^+(X) \cap F \cap Post^\star(\ell_0)) \;=\; \emptyset.$$

Remark that, noting $E_0 = Post^\star(H_0) \cap \mathcal{F}$ and $E_i = Post^+(E_{i-1}) \cap \mathcal{F}$, at the end of the i-st passage in the while loop of Algorithm 10, $C = E_i$.

Thanks to Theorem 5.37, we only need to prove the following lemma to ensure the correctness of Algorithm 10.

Lemma 5.38. *Let $i \geqslant 1$. $E_i = E_{i-1}$ iff $E_{i-1} = GFP(\lambda X.Post^+(X) \cap \mathcal{F} \cap Post^\star(H_0))$.*

Proof. ($\Leftarrow$) Suppose that $E_{i-1} = GFP(\lambda X.Post^+(X) \cap \mathcal{F} \cap Post^\star(H_0))$. In particular, $E_{i-1} = Post^+(E_{i-1}) \cap \mathcal{F} \cap Post^\star(H_0)$, i.e.:

$$E_{i-1} = E_i \cap Post^\star(H_0) \quad (*).$$

We will show that, $\forall j \geqslant 1$, $E_j \subseteq E_{j-1}$: in particular it means that $E_i \subseteq E_0 = Post^\star(H_0) \cap \mathcal{F} \subseteq Post^\star(H_0)$. It enables to conclude from $(*)$ that $E_{i-1} = E_i$.
<u>Basis:</u> We prove that $E_1 \subseteq E_0$. We know that $Post^\star(H_0) \cap \mathcal{F} \subseteq Post^\star(H_0)$. As function $Post^+$ is monotonic: $Post^+(Post^\star(H_0) \cap \mathcal{F}) \subseteq Post^+(Post^\star(H_0)) \subseteq Post^\star(H_0)$. So, $E_1 = Post^+(Post^\star(H_0) \cap \mathcal{F}) \cap \mathcal{F} \subseteq Post^\star(H_0) \cap \mathcal{F} = E_0$.
<u>Induction:</u> Suppose that $\forall 0 < k < i$, $E_k \subseteq E_{k-1}$. We must prove that $E_i \subseteq E_{i-1}$. By induction hypothesis, $E_{i-1} \subseteq E_{i-2}$, and as function $Post^+$ is monotonic: $Post^+(E_{i-1}) \subseteq Post^+(E_{i-2})$. Hence, $E_i = Post^+(E_{i-1}) \cap \mathcal{F} \subseteq Post^+(E_{i-2}) \cap \mathcal{F} = E_{i-1}$.
($\Rightarrow$) Suppose that $E_i = E_{i-1}$. Let us first prove that E_{i-1} is a fixed point of $\lambda(X) = Post^+(X) \cap \mathcal{F} \cap Post^\star(H_0)$. $\lambda(E_{i-1}) = Post^+(E_{i-1}) \cap \mathcal{F} \cap Post^\star(H_0) = E_i \cap Post^\star(H_0) = E_i = E_{i-1}$.
Now, let us prove that E_{i-1} is a greatest fixed point of λ. Let $Y \supseteq E_{i-1}$ such that $Y = \lambda(Y)$. We must prove that $Y = E_{i-1}$. As we already know that $E_{i-1} \subseteq Y$, it remains to prove that $Y \subseteq E_{i-1}$. As $Y = \lambda(Y)$, we have $Y = Post^+(Y) \cap \mathcal{F} \cap Post^\star(H_0) \subseteq \mathcal{F} \cap Post^\star(H_0) = E_0$. As $Y \subseteq E_0$ and function $Post^+$ is monotonic, $Post^+(Y) \subseteq Post^+(E_0)$. So, $Y = Post^+(Y) \cap \mathcal{F} \cap Post^\star(H_0) \subseteq Post^+(E_0) \cap \mathcal{F} \cap Post^\star(H_0) = E_1 \cap Post^\star(H_0) = E_1$. Applying inductively the same reasoning, we obtain that, $\forall k \geqslant 0$, $Y \subseteq E_k$. Hence, $Y \subseteq E_{i-1}$. $\square$

Here is a theorem giving a rough size of the number of words that Algorithm 10 must explore in the worst case.

Theorem 5.39. *For every MITL formula Φ, the associated OCATA $\mathcal{A}_{\neg\Phi} = (\Sigma, L, \ell_0, F, \delta)$, and every timed automaton $\mathcal{B} = (\Sigma, B, b_0, X, F^{\mathcal{B}}, \delta^{\mathcal{B}})$ with n clocks, $\mathcal{H}$ (constructed thanks to $\mathcal{S}_{\mathcal{B},\neg\Phi}$) contains $O(2^m)$ elements, where $m = (|B| + |L|) \,.\, (2.c_{\max} + 2) \,.2.\, \max\left(\max_{i=1}^{n}(\frac{\max_i}{2}), n\right) . (M(\neg\Phi) + n)$.*

Proof. Let us note $m' = (|B| + |L|) \,.\, (2.c_{\max} + 2) \,.2.\, \max\left(\max_{i=1}^{n}(\frac{\max_i}{2}), n\right)$. There is $2^{m'}$ elements in $\Lambda = (B \cup L) \times REG \times \{\top, \bot\} \times \{1, 2, \ldots, \max(\max_{i=1}^{n}(\frac{\max_i}{2}), n)\}$. $\mathcal{H}$ is a set of words of $\Lambda^\star$ having at most $M(\neg\Phi) + n$ letters. There are: 1 word of 0 letters over Λ ; $2^{m'}$ words of 1 letters over Λ ; $2^{2.m'}$ words of 2 letters over Λ ; $2^{3.m'}$ words of 3 letters over Λ ; ... ; $2^{(M(\neg\Phi)+n).m'}$ words of $M(\neg\Phi) + n$ letters over Λ. Globally, there are $\Sigma_{j=0}^{M(\neg\Phi)+n} 2^{j.m'} = O\left(2^{(M(\neg\Phi)+n).m'}\right) = O(2^m)$ words in $\mathcal{H}$. □

5.5 Zone-based algorithm

As for the finite word setting, the aim of this section is to provide a heuristic of the previously presented algorithms, over infinite words, using zones instead of regions. We thus show how zones for OCATA [2] can be adapted to represent set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$. Let us recall we fixed an MITL formula Φ, the OCATA $\mathcal{A}_{\neg\Phi} = (\Sigma, L, \ell_0, F, \delta)$ representing the negation of Φ and a TA $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$. As previously, a zone is an extended guard on the values of the clocks and clock copies, with additional information. This supplementary information aims to associate clock copies of $\mathcal{A}_{\neg\Phi}$ and clocks of $\mathcal{B}$ to the locations they are in, and to Miyano-Hayashi markers. All the results presented in this section are twins of those of Section 4.4. The unique difference is the treatment needed to cope with the Miyano-Hayashi markers.

In the sequel, we use again the following notations from Section 4.4:

- x is the unique clock of $\mathcal{A}_{\neg\Phi}$ and $x_1^{\mathcal{B}}, x_2^{\mathcal{B}}, \ldots, x_n^{\mathcal{B}}$ the clocks of $\mathcal{B}$.
- $Copies = \{x_1, x_2, \ldots, x_{M(\neg\Phi)/2}, y_1, y_2, \ldots, y_{M(\neg\Phi)/2}\}$ is the set of $M(\neg\Phi)$ copies of x: each pair of clock copies (x_i, y_i) will represent an interval.
- $Copies_{begin} := \{x_1, x_2, \ldots, x_{M(\neg\Phi)/2}\}$, $Copies_{end} := \{y_1, y_2, \ldots, y_{M(\neg\Phi)/2}\}$ and for $1 \leqslant m \leqslant M(\neg\Phi)/2$: $Copies^m := \{x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_m\}$ and $Copies^m_{begin} := \{x_1, x_2, \ldots, x_m\}$, while $Copies^0(x) = Copies^0_{begin} = \varnothing$.

Once again, the definition of zone presented below uses a supplementary clock x_0 whose value is always 0 and is based on the notion of extended guard (see Definition 4.58). It is very close to the definition of zone given in Definition 4.59, over finite words. However, this definition not only takes care of giving, for each clock copy of $\mathcal{A}_{\neg\Phi}$ (and for the timed automaton $\mathcal{B}$), the location it is in, but it also precises the Miyano-Hayashi marker associated to each copy of clock of $\mathcal{A}_{\neg\Phi}$ (and associated to $\mathcal{B}$). These informations are crucial in way to perform model-checking over $\mathcal{A}_{\neg\Phi}$ and $\mathcal{B}$.

Definition 5.40. *A zone $\mathcal{Z}_m$ is a tuple $(loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ where:*

- $m \in \mathbb{N}_0$,
- $loc_{\mathcal{A}} : Copies^m_{begin} \to L \times \{\top, \bot\}$,
- $loc_{\mathcal{B}} \in \mathcal{B} \times \{\top, \bot\}$ *is a pair consisting in a location of B and a marker in* $\{\top, \bot\}$, *and*
- Z *is an extended guard on* $Copies^m \cup X \cup \{x_0\}$ *(it is a 'classical zone' on this set of clocks [27]).*

A zone $\mathcal{Z}_m$ represents a particular set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ given by the following definition:

Definition 5.41. *Let $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ be a zone on set of clocks $Copies^m \cup X = \{x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_m\} \cup \{x_1^{\mathcal{B}}, x_2^{\mathcal{B}}, \ldots, x_n^{\mathcal{B}}\}$, such that $loc_{\mathcal{B}} = (loc, mark)$.*

Then, we let $[\![\mathcal{Z}_m]\!]$ *be the denotation of* $\mathcal{Z}_m$ *defined as the following set of states of* $\mathcal{S}_{\mathcal{B},\neg\Phi}$*:*

$$\{ \ (loc, v(x_1^{\mathcal{B}}), v(x_2^{\mathcal{B}}), \ldots, v(x_n^{\mathcal{B}}), mark), \ (loc^{x_1}, [v(x_1), v(y_1)], mark^{x_1}), \ \ldots, \\ (loc^{x_m}, [v(x_m), v(y_m)], mark^{x_m}) \ \},$$

such that v *is a valuation of* $Copies^m \cup X$ *with* $v \models \mathcal{Z}_m$ *and* $(loc^c, mark^c) = loc_{\mathcal{A}}(c)$ *for all* $c \in Copies^m_{begin}$*.*
For a set ζ *of zones, we note* $[\![\zeta]\!] := \bigcup_{\mathcal{Z}_m \in \zeta} [\![\mathcal{Z}_m]\!]$*.*

By abuse of notation, we sometimes write $s \in \mathcal{Z}_m$ instead of $s \in [\![\mathcal{Z}_m]\!]$.

Remark 5.42. We notice that, in the definition of zone $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$, there is no assumption about the constants to use in the extended guards of Z. In particular, they may contain constants greater than the maximal constants of $\mathcal{A}_{\neg\Phi}$ and $\mathcal{B}$.

Definition 5.43. *The initial zone is* $\mathcal{Z}_1^{init} = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ *with* $Z = x_1^{\mathcal{B}} = 0 \wedge \cdots \wedge x_n^{\mathcal{B}} = 0 \wedge x_1 = 0 \wedge y_1 = 0$*,* $loc_{\mathcal{A}}(x_1) = (\ell_0, mark^{\mathcal{A}})$ *and* $loc_{\mathcal{B}} = (b_0, mark^{\mathcal{B}})$ *where* $mark^{\mathcal{A}} = \top$ *iff* $\ell_0 \in F$ *and* $mark^{\mathcal{B}} = \top$ *iff* $b_0 \in F^{\mathcal{B}}$*.*
A zone $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ *is accepting iff* $\forall 1 \leqslant i \leqslant m$*,* $\exists \ell_i \in L$ *such that* $loc_{\mathcal{A}}(x_i) = (\ell_i, \top)$ *and* $\exists b \in B$ *such that* $loc_{\mathcal{B}} = (b, \top)$*.*

Example 5.44. Let us consider again the TA $\mathcal{B}$ of Figure 4.12 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 5.6. The initial zone is $\mathcal{Z}_1^{init} = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ with:

- $loc_{\mathcal{A}}(x_1) = (\ell_\Box, \top)$,
- $loc_{\mathcal{B}} = (b_0, \top)$, and
- $Z = x_1^{\mathcal{B}} = 0 \ \wedge \ x_1 = 0 \ \wedge \ y_1 = 0$.

The unique state of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ represented by $\mathcal{Z}_1^{init}$ is:

$$\{(b_0, 0, \top), (\ell_\Box, [0, 0], \top)\},$$

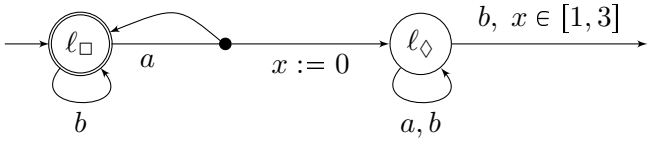


Figure 5.6: OCATA $\mathcal{A}_{\neg\Phi}$ with $\neg\Phi \equiv \Box(a \Rightarrow \Diamond_{[1,3]} b)$.

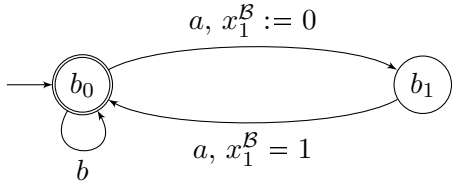


Figure 5.7: A timed automaton $\mathcal{B}$.

it is the initial state of $\mathcal{S}_{\mathcal{B},\neg\Phi}$. $\mathcal{Z}_1^{init}$ is accepting because the markers of $loc_{\mathcal{A}}(x_1)$ and $loc_{\mathcal{B}}$ are both $\top$.
Here is an example of a zone using 2 pairs of clock copies: $\mathcal{Z}_2 := (loc'_{\mathcal{A}}, loc'_{\mathcal{B}}, Z')$, with:

- $loc'_{\mathcal{A}}(x_1) = (\ell_\Box, \top)$ and $loc'_{\mathcal{A}}(x_2) = (\ell_\Diamond, \bot)$
- $loc'_{\mathcal{B}} = (b_1, \bot)$, and
- $Z' = x_1 \geqslant 0 \wedge y_1 \geqslant 0 \wedge x_1 = y_1 \wedge x_2 = 0 \wedge y_2 = 0 \wedge x_1^{\mathcal{B}} = 0$.

$\mathcal{Z}_2$ represents the set of states:

$$\{(b_1, 0, \bot), (\ell_\Box, [t,t], \top), (\ell_\Diamond, [0,0], \bot) \mid t \in \mathbb{R}^+\}.$$

As for the finite words setting, our algorithm on zones will start from the initial zone and compute its successors. But this time, we are looking for an *accepting* zone *reachable* from the initial zone and *reachable from itself*. We now define the timed and discrete successors of a given zone.

Definition 5.45. *Let $\mathcal{Z}_m$ be a zone. $Post_T(\mathcal{Z}_m)$ denotes the zone such that:*

$$[\![Post_T(\mathcal{Z}_m)]\!] = \{s' \in \mathcal{S}_{\mathcal{B},\neg\Phi} \mid \exists s \in \mathcal{Z}_m \text{ and } t \in \mathbb{R}^+ \text{ such that } s \overset{t}{\rightsquigarrow} s'\}.$$

$Post_T(\mathcal{Z}_m)$ can be computed from $\mathcal{Z}_m$ using the same method as for finite words (see Definition 4.64).

Definition 5.46. *Let $\mathcal{Z}_m$ be a zone. $Post_D(\mathcal{Z}_m)$ denotes a set of zones $\mathcal{Z}$ such that:*

$$[\![Post_T(\mathcal{Z}_m)]\!] = \{s' \in \mathcal{S}_{\mathcal{B},\neg\Phi} \mid \exists s \in \mathcal{Z}_m \text{ and } \sigma \in \Sigma \text{ such that } s \xrightarrow{\sigma} s'\}.$$

Remark 5.47. As for the finite words setting, we note that a state s of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ and one of its discrete successors do not necessarily use the same number of clock copies: an element of $Post_D(\mathcal{Z}_m)$ may be a zone using a number of pairs of clock copies different from m.

We start giving two examples of computations of $Post_D(\mathcal{Z}_m)$ before to give an intuition on the general way to determine it. Finally, we will formally define how to compute it.

Example 5.48. Let us consider again the TA $\mathcal{B}$ of Figure 5.7 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 5.6. We consider the accepting zone $\mathcal{Z}_1 := (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ with:

- $loc_{\mathcal{A}}(x_1) = (\ell_\Box, \top)$,
- $loc_{\mathcal{B}} = (b_0, \top)$, and
- $Z = x_1^{\mathcal{B}} \geqslant 0 \wedge x_1 \geqslant 0 \wedge y_1 \geqslant 0 \wedge x_1 = y_1 \wedge x_1 = x_1^{\mathcal{B}}$.

We are looking for $Post_D(\mathcal{Z}_1)$. Let us consider $s \in \mathcal{Z}_1$. s is of the form:

$$\{(b_0, t, \top), (\ell_\Box, [t, t], \top)\}, \text{ for some } t \in \mathbb{R}^+.$$

On the one hand, a discrete successor of such an s, reading $b \in \Sigma$, will be a state of the form:

$$\{(b_0, t, \top), (\ell_\Box, [t, t], \top)\}.$$

Indeed, even if s is accepting and the marking of locations must be started again, b_0 and $\ell_\Box$ are also accepting locations. This set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is exactly represented by the zone $\mathcal{Z}_1$ itself. On the other hand, a discrete successor of such an accepting s, reading $a \in \Sigma$, will be a state of the form:

$$\{(b_1, 0, \bot), (\ell_\Box, [t, t], \top), (\ell_\Diamond, [0, 0], \bot)\}, \text{ for } t \in \mathbb{R}^+.$$

This set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ is exactly represented by the zone $\mathcal{Z}_2^a := (loc'_{\mathcal{A}}, loc'_{\mathcal{B}}, Z')$, with:

- $loc'_{\mathcal{A}}(x_1) = (\ell_\Box, \top)$ and $loc'_{\mathcal{A}}(x_2) = (\ell_\Diamond, \bot)$
- $loc'_{\mathcal{B}} = (b_1, \bot)$, and
- $Z' = x_1 \geqslant 0 \wedge y_1 \geqslant 0 \wedge x_1 = y_1 \wedge x_2 = 0 \wedge y_2 = 0 \wedge x_1^{\mathcal{B}} = 0$.

We conclude that $Post_D(\mathcal{Z}_1) := \{\mathcal{Z}_1, \mathcal{Z}_2^a\}$.

Example 5.49. Let us still consider the TA $\mathcal{B}$ of Figure 5.7 and the OCATA $\mathcal{A}_{\neg\Phi}$ of Figure 5.6. We consider the zone $\mathcal{Z}_2 := (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ with:

- $loc_{\mathcal{A}}(x_1) = (\ell_\Box, \top)$ and $loc_{\mathcal{A}}(x_2) = (\ell_\Diamond, \bot)$,
- $loc_{\mathcal{B}} = (b_1, \bot)$, and
- $Z = x_1 \geqslant 1 \wedge y_1 \geqslant 1 \wedge x_1 = y_1 \wedge x_2 = 1 \wedge y_2 = 1 \wedge x_1^{\mathcal{B}} = 1$.

We are looking for $Post_D(\mathcal{Z}_2)$. Let us consider $s \in \mathcal{Z}_2$. s is of the form:

$$\{(b_1, 1, \bot), (\ell_\Box, [t, t], \top), (\ell_\Diamond, [1, 1], \bot)\}, \text{ for some } t \geqslant 1.$$

Remark that we can only read an a from such a state (see the unique arc starting from b_1 in $\mathcal{B}$). A discrete successor of s can either be of the form:

(i) $\{(b_0, 1, \top), (\ell_\Box, [t, t], \top), (\ell_\Diamond, [0, 0], \bot), (\ell_\Diamond, [1, 1], \bot)\}$, or

(ii) $\{(b_0, 1, \top), (\ell_\Box, [t, t], \top), (\ell_\Diamond, [0, 1], \bot)\}$.

We note that case (i) is only possible because $M(\neg\Phi)$ is big enough. The set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ presented in (i) is exactly represented by the zone $\mathcal{Z}_3 := (loc^3_{\mathcal{A}}, loc^3_{\mathcal{B}}, Z^3)$ with:

- $loc^3_{\mathcal{A}}(x_1) = (\ell_\Box, \top)$, $loc^3_{\mathcal{A}}(x_2) = (\ell_\Diamond, \bot)$ and $loc^3_{\mathcal{A}}(x_3) = (\ell_\Diamond, \bot)$,
- $loc^3_{\mathcal{B}} = (b_0, \top)$, and
- $Z^3 = x_1 \geqslant 1 \wedge y_1 \geqslant 1 \wedge x_1 = y_1 \wedge x_2 = 0 \wedge y_2 = 0 \wedge x_3 = 1 \wedge y_3 = 1 \wedge x_1^{\mathcal{B}} = 1$.

The set of states of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ presented in (ii) is exactly represented by the zone $\mathcal{Z}_2^{group} := (loc^{group}_{\mathcal{A}}, loc^{group}_{\mathcal{B}}, Z^{group})$ with:

- $loc^{group}_{\mathcal{A}}(x_1) = (\ell_\Box, \top)$ and $loc^{group}_{\mathcal{A}}(x_2) = (\ell_\Diamond, \bot)$,
- $loc^{group}_{\mathcal{B}} = (b_0, \top)$, and
- $Z^{group} = x_1 \geqslant 1 \wedge y_1 \geqslant 1 \wedge x_1 = y_1 \wedge x_2 = 0 \wedge y_2 = 1 \wedge x_1^{\mathcal{B}} = 1$.

We conclude that $Post_D(\mathcal{Z}_2) := \{\mathcal{Z}_3, \mathcal{Z}_2^{group}\}$.

In general, we compute $Post_D(\mathcal{Z}_m)$, where $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$, in a similar way as over finite words, except that the Miyano-Hayashi markers must be kept updated. From a non-accepting zone, we proceed as follows.

a. When an interval stays in a same location ℓ, and is not merged with a new created interval in ℓ:
   - its marker stays unchanged if ℓ is non accepting, and
   - its marker becomes $\top$ if ℓ is accepting (this case is necessary when we reached a final zone and that all the markers were changed to $\bot$).

b. When a new interval $[0,0]$ is created in a location ℓ, it is because several intervals follows arcs leading to ℓ:

   - if all these intervals were marked by $\top$, this new interval is also marked by $\top$, and
   - in contrary, if at least one of these intervals was marked by $\bot$, this interval witnesses a branch which have not visited F yet. We must mark this new interval by $\bot$ to recall it.

c. When a new interval is created in a location ℓ (because several intervals follows arcs leading to ℓ) and merged with the smallest interval which were associated to ℓ, the interval to marked takes marker $\top$ iff:

   - ℓ is accepting, **or**
   - all the intervals arriving in ℓ and the previous smallest interval associated to ℓ are marked by $\top$.[5]

From an accepting zone, we start changing all the markers to $\bot$ before to proceed as previously explained.

In the sequel, we formally define what we have just intuitively explained. Let us fix a zone $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$. We will use the following formal notations:

- for an arc t of $\mathcal{A}_{\neg\Phi}$ of the form $(\ell, \sigma, \bigwedge_k A_{j,k})$ such that $\bigwedge_k A_{j,k}$ is a disjunct in $\delta(\ell, \sigma)$, we let $\mathsf{dest}(t) = \{\ell \mid \exists k : A_{j,k} = \ell \vee A_{j,k} = x.\ell\}$ to be the set of destinations of t.
- We first collect the arcs labelled by σ that start in the current locations of both automata and combine them:

  1. $\forall c \in Copies^m$, $\forall \sigma \in \Sigma$:
     $E(c, \sigma) := \{$ arcs labelled by σ and whose starting location is the location of $loc_{\mathcal{A}}(c)\}$,

[5] In all the other cases, a marker $\bot$ should recall that at least one branch leading to the presently considered interval has not visited F yet.

2. $\forall \sigma \in \Sigma$:
   $E(\mathcal{B}, \sigma) := \{$arcs starting from the location of $loc_{\mathcal{B}}$ and labelled by $\sigma\}$,
3. $\forall \sigma \in \Sigma$:
   $Z \odot \sigma := E(\mathcal{B}, \sigma) \times \prod_{i=1}^{m} E(x_i, \sigma)$.

- Then, we introduce definitions that will allow us to select from the zone the valuations that satisfy the guards of selected arcs from $\mathcal{A}_{\neg\Phi}$ and $\mathcal{B}$. To do this, we collect the guards of all these arcs and combine them:

  1. for every arc t:
     $Constr(t) := \{c \mid c$ is a clock constraint in the guard of $t\}$,
  2. for $t_{\mathcal{B}} = (b_{start}, \sigma, g, r, b_{arrival})$ an arc of $\mathcal{B}$, and for a sequence $t_1, \ldots, t_m$ of arcs of $\mathcal{A}_{\neg\Phi}$, we let:

$$g_{t_{\mathcal{B}}, t_1, \ldots, t_m} := g \quad \wedge \bigwedge_{\substack{1 \leqslant i \leqslant m \\ c_i \in Constr(t_i)}} \left(c_i|_{x=x_i} \wedge c_i|_{x=y_i} \right).$$

- Now, we fix an arc $t_{\mathcal{B}} = (b_{start}, \sigma, g, r, b_{arrival})$ of $\mathcal{B}$, and a sequence $t_1, \ldots, t_m$ of arcs of $\mathcal{A}_{\neg\Phi}$ to be fired simultaneously. We introduce definitions that allow us to compute the locations, the clock copies present in them, and the associated Miyano-Hayashi markers that will be active after firing these arcs. We need to distinguish between locations on which we loop from others. Indeed, when we do not loop, we must reset one (see Example 5.49, point (ii) and zone $\mathcal{Z}_3$) or two (see Example 5.49, point (i) and zone $\mathcal{Z}_2^{group}$) clock(s) in the destination location.

  1. We first define the following set keeping the destination location and marker of $\mathcal{B}$, and the destination location and the marker of each clock copy of $\mathcal{A}_{\neg\Phi}$ that changes of location:

$$\begin{aligned} LOCM(t_{\mathcal{B}}, t_1, \ldots, t_m) \ := & \\ & \{(b_{arrival}, mark) \mid mark \text{ is the marker of } loc_{\mathcal{B}}\} \\ & \cup \{ (loc, mark) \mid \exists 1 \leqslant i \leqslant m : loc \in \mathsf{dest}(t_i) \backslash \{loc_{\mathcal{A}}(x_i)\} \text{ and} \\ & \qquad mark \text{ is the marker of } loc_{\mathcal{A}}(x_i)\}. \end{aligned}$$

2. We define a set containing the clock copies that loop on their location:

$$\begin{aligned} Loop(t_1,\ldots,t_m) \;&:= \{x_i \mid loc_{\mathcal{A}}(x_i) = (\ell, mark) \text{ and } \ell \in \mathsf{dest}(t_i)\} \\ &\cup \{y_i \mid loc_{\mathcal{A}}(x_i) = (\ell, mark) \text{ and } \ell \in \mathsf{dest}(t_i)\}. \end{aligned}$$

3. From $Loop(t_1,\ldots,t_m)$, we need to extract the clock copies with the minimal value associated to each location ℓ on which some clock copies loop. Indeed, in case of a merging of intervals in ℓ, such a clock copy will be reset (see Example 5.49, point (i) and zone $\mathcal{Z}_2^{group}$):

$$\begin{aligned} &minLoop(t_1,\ldots,t_m) \;:= \\ &\quad \{x_i \in Loop(t_1,\ldots,t_m) \cap Copies_{begin} \mid \\ &\quad \forall x_i' \in Loop(t_1,\ldots,t_m) \text{ with } loc_{\mathcal{A}}(x_i) = loc_{\mathcal{A}}(x_i'), \\ &\quad x_i' \geqslant x_i \text{ is implied by the extended guard of } Z_m\}. \end{aligned}$$

- Finally, to conclude the effect of the combined firing of $(t_{\mathcal{B}}, t_1, \ldots, t_m)$, we need to compute which clocks and clock copies will be reset. By the previous definitions, we only need to reset one or two clock copie(s) in the locations ℓ such that $(\ell, mark) \in LOCM(t_{\mathcal{B}}, t_1, \ldots, t_m)$ for some $mark \in \{\top, \bot\}$. For each such location ℓ, we note r^ℓ an element of the set $\{\{x^\ell\}, \{x, y\}\}$ such that:

  (a) $r^\ell = \{x^\ell\}$ implies that $x^\ell \in minLoop(t_1,\ldots,t_m)$ and $loc_{\mathcal{A}}(x^\ell) = \ell$, and

  (b) $r^\ell = \{x, y\}$ implies that $x \in Copies_{begin} \backslash Loop(t_1,\ldots,t_m)$ and $y \in Copies_{end} \backslash Loop(t_1,\ldots,t_m)$.

  we furthermore require that

  (c) for all ℓ_i, ℓ_j such that $(\ell_i, mark_i) \in LOCM(t_{\mathcal{B}}, t_1, \ldots, t_m)$ and $(\ell_j, mark_j) \in LOCM(t_{\mathcal{B}}, t_1, \ldots, t_m)$ for some $mark_i, mark_j \in \{\top, \bot\}$: $r^{\ell_i} \cap r^{\ell_j} = \varnothing$.

Thanks to those notations, we are now able to define the elements of $Post_D(\mathcal{Z}_m)$.

For $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$,

$$\mathcal{Z}'_{m'} = (loc'_{\mathcal{A}}, loc'_{\mathcal{B}}, Z') \in Post_D(\mathcal{Z}_m)$$
iff

either (1) $\mathcal{Z}_m$ is not accepting and:
there exists $\sigma \in \Sigma$, $(t_{\mathcal{B}}, t_1, \ldots, t_m) \in Z \odot \sigma$ and r^ℓ, for all ℓ such that $(\ell, mark) \in LOCM(t_{\mathcal{B}}, t_1, \ldots, t_m)$ for some $mark \in \{\top, \bot\}$, such that $g_{t_{\mathcal{B}}, t_1, \ldots, t_m} \cap Z$ is satisfiable and:

- $Z' = (g_{t_{\mathcal{B}}, t_1, \ldots, t_m} \cap Z) \left[\left(r \cup \bigcup r^\ell\right) := 0\right]$;
- $\forall x_i \in Loop(t_1, \ldots, t_m) \setminus \left(\bigcup r^\ell\right)$[6], $loc'_{\mathcal{A}}(x_i)$ consists in the location of $loc_{\mathcal{A}}(x_i)$ and:
  - the marker $\top$ if the location of $loc_{\mathcal{A}}(x_i)$ is in F,
  - the marker of $loc_{\mathcal{A}}(x_i)$ otherwise;

  $\forall x_\ell \in \left(^\ell \cap Copies_{begin}\right) \setminus Loop(t_1, \ldots, t_m)$[7], $loc'_{\mathcal{A}}(x_\ell)$ consists in the location ℓ and the marker:
  - $\top$ iff $\ell \in F$ or $(\ell, \bot) \notin LOCM(t_{\mathcal{B}}, t_1, \ldots, t_m)$[8]

  $\forall x_\ell \in r^\ell \cap Loop(t_1, \ldots, t_m)$[9], $loc'_{\mathcal{A}}(x_\ell)$ consists in the location ℓ and the marker:
  - $\top$ iff $\ell \in F$ or $((\ell, \bot) \notin LOCM(t_{\mathcal{B}}, t_1, \ldots, t_m)$ and the marker of $loc_{\mathcal{A}}(x_\ell)$ is $\top$);
- $loc'_{\mathcal{B}}$ consists in the destination location $\ell^{\mathcal{B}}$ of $t_{\mathcal{B}}$ and the marker:
  - $\top$ iff $\ell^{\mathcal{B}} \in F^{\mathcal{B}}$ or $(\ell^{\mathcal{B}}, \top) \in LOCM(t_{\mathcal{B}}, t_1, \ldots, t_m)$;

[6] case a. of the previously given intuition.
[7] case b. of the previously given intuition.
[8] Several branches reach location ℓ with different markers and creat: if one of them was $\bot$, it witnesses a branch that has not yet visited F and this information must be retained using a $\bot$ marker.
[9] case c. of the previously given intuition.

or (2) $\mathcal{Z}_m$ is accepting and:
defining $loc^\star_{\mathcal{A}}$ such that $\forall 1 \leqslant i \leqslant m$, if $loc_{\mathcal{A}}(x_i) = (\ell, \top)$, then $loc^\star_{\mathcal{A}}(x_i) = (\ell, \bot)$, $\mathcal{Z}'_{m'}$ satisfies the conditions of case (1) in which $loc^\star_{\mathcal{A}}$ plays the role of $loc_{\mathcal{A}}$.

The following proposition states that the elements of $Post_D(\mathcal{Z}_m)$ were correctly defined. Its proof is close to that of Proposition 4.70 (its twin for the setting of finite words): it can be found in the appendix.

Proposition 5.50. Let $\mathcal{Z}_m$ be a zone.

$$[\![Post_D(\mathcal{Z}_m)]\!] = \{s' \mid \exists s \in \mathcal{Z}_m \text{ such that } s \to s' \text{ in } \mathcal{S}_{\mathcal{B},\neg\Phi}\}.$$

The operators $Post$, $Post^*$ and $Post^+$ are defined in a similar way for zones over infinite words and over finite words (see Definitions 4.71 and 4.72).

We recall that, over regions, as $\mathcal{H}$ is finite, when considering $\mathcal{W} \subseteq \mathcal{H}$, we had the existence of an $m \in \mathbb{N}$ such that: $Post^*(\mathcal{W}) = \bigcup_{n=1}^{m} Post^n(\mathcal{W})$ (see Definition 5.34). Althrough there is no maximal value bounding the constants present in the extended guards of a zone (see Remark 5.42), we recall (see Section 4.4) that this problem was subject to debate, a debate that was finally closed thanks to the paper [15]. The technique used consists in the approximation of each zone $\mathcal{Z}$ by $Approx_\beta(\mathcal{Z})$ (see Section 4.4 and [15] for further details). The number of such 'approximated' zones is then finite and a correct algorithm in the spirit of Algorithm 10 (over regions), will thus terminate. Hence, we can present Algorithm 11, that uses approximation $Approx_\beta$ and solves the MITL model-checking problem using the same outline as Algorithm 10. This algorithm terminates because there is only a finite number of 'approximated' zones to explore.

Algorithm 11 MITLModelCheckingOverInfiniteWordsWithZones

Input: A TA $\mathcal{B}$ and the ATA $\mathcal{A}_{\neg\Phi}$, for $\Phi \in$ MITL.

Output: 'true' iff $\mathcal{B} \models \Phi$.

1: $C \leftarrow \emptyset$
2: $D \leftarrow Approx_\beta(Post^\star(\mathcal{Z}_1^{init})) \cap \mathcal{F}$
3: **while** $C \neq D$ **do**
4: $\quad C \leftarrow D$
5: $\quad D \leftarrow Approx_\beta(Post^+(D)) \cap \mathcal{F}$
6: **end while**
7: **if** $D = \emptyset$ **then**
8: $\quad$ **return** true
9: **else**
10: $\quad$ **return** false
11: **end if**

CHAPTER 6

Experimental results

In this chapter, we give the results obtained when testing the program implementing our different algorithms on several benchmarks. In addition to the algorithms presented in the previous chapters, we implemented other versions, in way to avoid considering useless clock copies. We start expliciting the trick we uses to eliminate these useless clock copies before to present the obtained experimental results on all the versions of our algorithms.

Eliminating useless clock copies. In many practical examples, MITL formulas contain modalities of the form $U_{[0,+\infty[}$ or $\tilde{U}_{[0,+\infty[}$ that do not impose any real-time constraints (in some sense, they are LTL modalities). For instance, consider the $\Box$ modality in $\Phi = \Box(a \Rightarrow \Diamond_{[1,2]} b)$. When this occurs in a formula Φ, we can simplify the representation of configurations of $\mathcal{A}_\Phi$, by dropping the values of the clocks associated to those modalities (these clocks can be regarded as *inactive* in the sense of [26]). We call those configurations *reduced configurations*.

Example 6.1. For instance, let us consider the OCATA $\mathcal{A}_\Phi$ of Figure 6.1. Eliminating useless clock copies amounts to skipping the clocks associated to $\ell_\Box$. Let

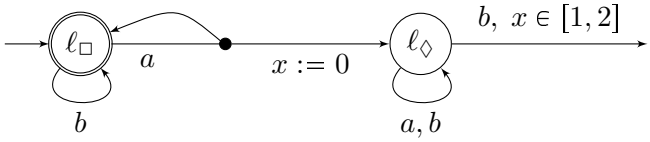


Figure 6.1: OCATA $\mathcal{A}$ with $L(\mathcal{A}) = [\![\, \Box(a \Rightarrow \Diamond_{[1,2]} b) \,]\!]$.

us consider the configuration

$$\{(\ell_\Box, 0.1), (\ell_\Diamond, 0)\}$$

of $\mathcal{A}_\Phi$. It can be more simply represented by the pair

$$(\{\ell_\Box\}, \{(\ell_\Diamond, 0)\}).$$

One can also verify that the values of the clock copies present in the initial location $(\neg\Phi)_{init}$ of $\mathcal{A}_{\neg\Phi}$ are not relevant. Let us note

$$Sub_{[0,+\infty)}(\neg\Phi) \;:=\; \{\Phi \in Sub(\neg\Phi) \mid \Phi = \Phi_{init} \text{ or the outermost operator of } \Phi \text{ is } U_{[0,+\infty[} \text{ or } \tilde{U}_{[0,+\infty[}\}.$$

Then, the states of $\mathcal{A}$ can either be

- a couple (ℓ, I) where $\ell \in L \backslash Sub_{[0,+\infty)}(\neg\Phi)$ and $I \in \mathcal{I}(\mathbb{R}^+)$, or
- a singleton $\ell \in Sub_{[0,+\infty)}(\neg\Phi)$.

A reduced configuration is then a pair (S, C) where $S \subseteq Sub_{[0,+\infty)}(\neg\Phi)$ and C is a configuration of $\mathcal{A}_{\neg\Phi}$ such that $\forall(\ell, v) \in C,\ \ell \notin Sub_{[0,+\infty)}(\neg\Phi)$. It is clear that all the results presented in the previous chapters still hold on reduced configurations and we also implemented the algorithms they contain with them: the region or zone abstraction is only used on $L \backslash Sub_{[0,+\infty)}(\neg\Phi)$.

Tests results. To evaluate the practical feasibility of our approach, we have implemented the region and zone-based algorithms (over finite and infinite words)

for model-checking and satisfiability in a prototype tool. To the best of our knowledge, this is the first implementation to perform MITL model-checking and satisfiability using an automata-based approach. We first consider a benchmark for the *satisfiability problem*, adapted from the literature on LTL [35] and consisting of six parametric formulas (with $k \in \mathbb{N}$ and $I \in \mathcal{I}(\mathbb{N}^{+\infty})$):

$$\begin{array}{rcl} E(k,I) & = & \bigwedge_{i=1,\dots,k} \Diamond_I \, p_i \\ A(k,I) & = & \bigwedge_{i=1,\dots,k} \Box_I \, p_i \\ Q(k,I) & = & \bigwedge_{i=1,\dots,k} (\Diamond_I p_i \vee \Box_I p_{i+1}) \\ U(k,I) & = & (\dots(p_1 U_I p_2) U_I \dots) U_I p_k \\ T(k,I) & = & p_1 \tilde{U}_I (p_2 \tilde{U}_I (p_3 \dots p_{k-1} \tilde{U}_I p_k) \dots) \\ R(k,I) & = & \bigwedge_{i=1,\dots,k} (\Box_I \, (\Diamond_I p_i) \vee \Diamond_I (\Box_I p_{i+1})) \end{array}$$

Our second benchmark evaluates the performance of our *model-checking tool*. We consider a family of timed automata $\mathcal{B}_k^{lift}$ that model a *lift*, parametrised by the number k of floors. A button can be pushed at each floor to call it. The alphabet contains a letter l_i for each floor i meaning the lift has been called at this floor. A button to send the lift at each floor is present in it: the alphabet contains a letter b_i for each floor i meaning that button i has just been pushed. The lift takes 1 second to go from a floor to the next one and to open/close its doors: it stays 1 second open between these opening/closure. Letter o_i (respectively c_i) signifies the lift opens (respectively closes) its doors at floor i ; letter p_i means the lift passes floor i without stopping. The lift goes up (respectively down) as long as it is called upper (respectively lower). When it is not called anywhere, it goes to the medium floor and stays opened there.

Example 6.2. For instance, Figure 6.2 gives a representation of $\mathcal{B}_2^{lift}$. For the sake of simplicity, we represent two similar edges (same starting and ending locations, same reset) carrying two different letters by a unique edge carrying these two letters. A location of this automaton is a 4-tuple $(n, direction, go, open?)$ where

- n is the number of the floor the cabin is present in, with $0 \leqslant n < 2$, 0

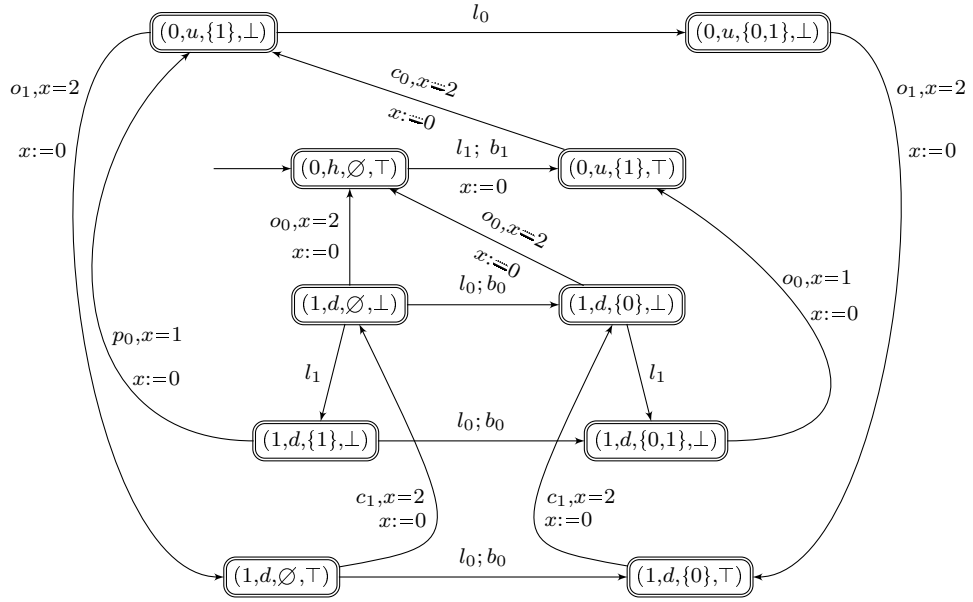


Figure 6.2: $\mathcal{B}_2^{lift}$

representing the ground floor,

- *direction* $\in \{u, d, h\}$ is u if the lift is going up, d if it is going down and h if it remains at the floor it is present in,

- *go* is the set of floors to which the cabin must go (because the lift has been called at this floor or because the button present in the cabin has been pushed to send the lift at this floor), and

- *open*? $\in \{\top, \bot\}$ is $\top$ iff the doors of the cabin are opened.

We implemented the algorithms of the LTL satisfiability and model-checking shown in the previous chapters in a java prototype. We performed our tests on a Mac Pro (mid 2010) with OS X Yosemite, processor 3.33 GHz, 6 core intel xeon, with a memory of 32 GB 1333 MHz DDR3 ECC. We used Java SE Runtime Environment (build 1.6.0_65-b14-466.1-11M4716).

Tables 6.1, 6.3, 6.2 and 6.4 are dedicated to the setting of *finite words*. They report on results over four versions of the satisfiability and model-checking algorithms:

- column 'Regions' is dedicated to the algorithm over regions (for the model-checking, see Algorithm 3 when the order is not used and Algorithm 8 when it is used),
- column 'Reduced regions' is dedicated to the algorithm over the reduced regions presented in the previous paragraph,
- column 'Zones' is dedicated to the algorithm over zones (for the model-checking, see Algorithm 9 when the order is not used),
- column 'Reduced zones' is dedicated to the algorithm over zones reduced in a way similar than regions.

Those columns contains the running time followed by the number of explored regions or zones. Those four tables also precise, for each tested formula, the expected[1] answer in column 'Sat ?', for the satisfiability benchmark, or in column 'ok ?', for the model-checking benchmark. The columns 'size' and 'OCATA/ TA size' show the very small size of the OCATA representing the formulas. A time out was set after 5 minutes, and OOM stands for 'out of memory'.

Tables 6.1 and 6.2 report on results of the prototype when performing satisfiability over the formulas of our first benchmark. Table 6.1 consists in the results obtained without using the order $\sqsubseteq$ of Definition 4.38, while Table 6.2 shows results obtained using this order.

Let us first consider, in Tables 6.1 and 6.2, the results concerning regions and reduced regions. As expected, the running times and number of explored regions are globally better over formulas containing intervals $[0, +\infty[$ when *reduced* regions are used. In average, over those formulas, from the use of regions to that

[1] and obtained, when the program terminates on this test !

of reduced regions, the number of explored regions is divided by 2 and the execution time is often divided by 5. However, this running time is divided by about 32 over formulas $U(10, [0, +\infty[)$ (with and without the use of the order) and by 49 over formula $R(5, [0, +\infty[)$ with the use of the order. The relation between zones and reduced zones is less pronounced: the numbers of explored zones and the running times are rather variable. Sometimes, the use of reduced zones gives better results, sometimes the opposite happens.

From the use of regions to that of zones, in average, the running time is divided by 6 and the number of explored regions/zones divided by 2.

When comparing results obtained without (see Table 6.1) and with (see Table 6.2) the use of the order, the execution times are rather similar, although the number of explored regions/zones is in average divided by 3 thanks to the use of the order. The similarity in the obtained running times certainly come from the supplementary computations necessary to the use of the order in our program.

Table 6.1: Finite words without order - Benchmark for the satisfiability. Reported values are execution time in ms / number of explored regions or zones.

Sat ?	Formula	Size	Regions	Reduced regions	Zones	Reduced zones
Sat	$E(5, [0, +\infty[)$	6	72 / 61	15 / 31	52 / 36	34 / 31
Sat	$E(10, [0, +\infty[)$	11	2,651 / 2,045	348 / 1,023	1,259 / 1,033	2,505 / 1,023
Sat	$E(5, [5, 8[)$	6	373 / 228	385 / 228	79 / 33	79 / 33
Sat	$E(10, [5, 8[)$	11	64,010 / 7,172	84,502 / 7,172	1,834 / 1,025	2,388 / 1,025
Unsat	$A(10, [0, +\infty[)$	11	2 / 1	1 / 1	4 / 1	4 / 1
Sat	$A(10, [5, 8[)$	11	393 / 1	41 / 1	37 / 1	38 / 1
Sat	$U(10, [0, +\infty[)$	10	255 / 8	8 / 7	15 / 1	5 / 1
Unsat	$U(2, [5, 8[)$	2	14 / 7	7 / 3	4 / 2	4 / 2
Unsat	$U(3, [5, 8[)$	3	OOM	OOM	221 / 63	222 / 63
Unsat	$U(4, [5, 8[)$	3	OOM	OOM	OOM	OOM
Sat	$T(10, [0, +\infty[)$	10	9 / 1	1 / 1	8 / 1	5 / 1
Sat	$T(10, [5, 8[)$	10	21 / 1	3 / 1	5 / 1	5 / 1
Sat	$R(5, [0, +\infty[)$	21	3,589 / 1,641	73 / 81	1,616 / 1,033	107 / 81
Sat	$R(10, [0, +\infty[)$	41	$>$ 5min	5,703 / 5,121	OOM	32,608 / 5,121
Sat	$R(5, [5, 8[)$	21	271 / 2	110 / 2	156 / 9	170 / 9
Sat	$R(10, [5, 8[)$	41	19,652 / 1	$>$ 5min	$>$ 5min	$>$ 5min
Sat	$Q(5, [0, +\infty[)$	11	45 / 39	10 / 20	38 / 29	18 / 20
Sat	$Q(10, [0, +\infty[)$	21	1,167 / 1,041	291 / 521	722 / 540	853 / 521
Sat	$Q(5, [5, 8[)$	11	243 / 12	123 / 10	87 / 16	88 / 16
Sat	$Q(10, [5, 8[)$	21	84,229 / 849	11,222 / 302	$>$ 5min	$>$ 5min

Table 6.2: Finite words with order - Benchmark for the satisfiability. Reported values are execution time in ms / number of explored regions or zones.

Sat ?	Formula	Size	Regions	Reduced regions	Zones	Reduced zones
Sat	$E(5, [0, +\infty[)$	6	110 / 21	21 / 11	47 / 11	26 / 11
Sat	$E(10, [0, +\infty[)$	11	2,749 / 505	387 / 253	973 / 253	847 / 253
Sat	$E(5, [5, 8[)$	6	527 / 81	562 / 81	81 / 12	85 / 12
Sat	$E(10, [5, 8[)$	11	69,097 / 1,775	70,794 / 1,775	1,682 / 254	2,207 / 254
Unsat	$A(10, [0, +\infty[)$	11	2 / 1	2 / 1	4 / 1	4 / 1
Sat	$A(10, [5, 8[)$	11	398 / 1	41 / 1	37 / 1	39 / 1
Sat	$U(10, [0, +\infty[)$	10	223 / 2	6 / 2	14 / 1	5 / 1
Unsat	$U(2, [5, 8[)$	2	15 / 7	7 / 3	4 / 2	4 / 2
Unsat	$U(3, [5, 8[)$	3	177,713 / 2,434	116,314 / 2,434	28 / 4	29 / 4
Unsat	$U(4, [5, 8[)$	3	$>$ 5min	$>$ 5min	$>$ 5min	OOM
Sat	$T(10, [0, +\infty[)$	10	8 / 1	1 / 1	9 / 1	5 / 1
Sat	$T(10, [5, 8[)$	10	21 / 1	3 / 1	4 / 1	5 / 1
Sat	$R(5, [0, +\infty[)$	21	1,824 / 378	127 / 66	1,943 / 207	115 / 66
Sat	$R(10, [0, +\infty[)$	41	OOM	9,892 / 4,610	$>$ 5min	40,801 / 4,610
Sat	$R(5, [5, 8[)$	21	270 / 2	109 / 2	157 / 9	171 / 9
Sat	$R(10, [5, 8[)$	41	18,101 / 1	$>$ 5min	$>$ 5min	$>$ 5min
Sat	$Q(5, [0, +\infty[)$	11	66 / 14	13 / 8	40 / 10	16 / 8
Sat	$Q(10, [0, +\infty[)$	21	1,251 / 253	320 / 127	645 / 127	475 / 127
Sat	$Q(5, [5, 8[)$	11	246 / 12	123 / 10	86 / 16	87 / 16
Sat	$Q(10, [5, 8[)$	21	16,816 / 127	5,926 / 223	34,058 / 347	36,514 / 347

Tables 6.3 and 6.4 give the model-checking results obtained on our prototype tool, when considering the timed automaton $\mathcal{B}_n^{lift}$, representing a lift with n floors (see the first column), and the formula given in the second column.
Table 6.3 presents results obtained without using the order $\sqsubseteq$ of Definition 4.38, while Table 6.4 displays results obtained using it.

Let us now consider the results over regions and reduced regions in Tables 6.3 and 6.4. The running times and numbers of explored regions are globally similar and sometimes the use of reduced regions leads to greater execution times, probably due to the supplementary computations necessary to maintain those reduced regions. The relation between zones and reduced zones is this time more pronounced: the running times and numbers of explored zones is reduced by 25% or by 50%, depending on the considered test.
When comparing the use of regions and that of zones in Tables 6.3 and 6.4, we see that zones enable to conclude a lot of tests on which the regions did not give any answer by 5 minutes. On the simplest examples, on which both regions and zones give an answer by 5 minutes, the running time and number of explored regions/zones is in average divided by 10 from the use of regions to that of zones. Then, the algorithm over regions quickly struggles while that over zones carries on answering by 5 minutes.
Let us still compare the results obtained without the use of the order (see Table 6.3) and with it (see Table 6.4). When considering formulas over a lift with two floors, the running times and numbers of explored regions/zones are globally similar. Then, when considering zones, the use of the order turns out to be more and more efficient with the rising number of floors: when considering a lift with four or five floors, the running times are in average reduced by 30% and the numbers of explored zones are in average reduced by 50%, when using the order.

Table 6.3: Finite words without order - Benchmark for the model-checking. Reported values are execution time in ms / number of explored regions or zones.

Floors	Formula	OCATA/ TA size	OK ?	Regions	Reduced regions	Zones	Reduced zones
2	$\Box \bigwedge_{i=1,2} ((o_i \wedge \Diamond_{[0,+\infty[} c_i) \Rightarrow \Diamond_{]1,2]} c_i)$	6 / 10	×	60 / 55	57 / 52	20 / 23	14 / 21
2	$\Box \bigwedge_{i=1,2} ((b_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,4]} o_i)$	6 / 10	✓	201 / 224	221 / 215	22 / 31	13 / 25
2	$\Box \bigwedge_{i=1,2} ((l_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,6]} o_i)$	6 / 10	✓	583 / 690	630 / 669	41 / 52	23 / 41
3	$\Box \bigwedge_{i=1,\ldots,3} ((o_i \wedge \Diamond_{[0,+\infty[} c_i) \Rightarrow \Diamond_{]1,2]} c_i)$	8 / 37	×	239 / 253	289 / 243	91 / 90	72 / 79
3	$\Box \bigwedge_{i=1,\ldots,3} ((b_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,12]} o_i)$	8 / 37	✓	>5min	>5min	510 / 553	480 / 388
3	$\Box \bigwedge_{i=1,\ldots,3} ((l_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,14]} o_i)$	8 / 37	✓	>5min	>5min	601 / 816	573 / 570
4	$\Box \bigwedge_{i=1,\ldots,4} ((o_i \wedge \Diamond_{[0,+\infty[} c_i) \Rightarrow \Diamond_{]1,2]} c_i)$	10 / 114	×	649 / 722	763 / 705	508 / 259	386 / 237
4	$\Box \bigwedge_{i=1,\ldots,4} ((b_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,20]} o_i)$	10 / 114	✓	>5min	>5min	9,671 / 6,121	3,616 / 3,392
4	$\Box \bigwedge_{i=1,\ldots,4} ((l_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,22]} o_i)$	10 / 114	✓	>5min	OOM	16,124 / 8,345	6,406 / 4,721
5	$\Box \bigwedge_{i=1,\ldots,5} ((o_i \wedge \Diamond_{[0,+\infty[} c_i) \Rightarrow \Diamond_{]1,2]} c_i)$	12 / 311	×	2,185 / 1,804	3,535 / 1,715	1,005 / 605	1,198 / 567
5	$\Box \bigwedge_{i=1,\ldots,5} ((b_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,28]} o_i)$	12 / 311	✓	OOM	OOM	>5min	140,178 / 19,704
5	$\Box \bigwedge_{i=1,\ldots,5} ((l_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,30]} o_i)$	12 / 311	✓	OOM	OOM	>5min	282,504 / 25,368

Table 6.4: Finite words with order - Benchmark for the model-checking. Reported values are execution time in ms / number of explored regions or zones.

Floors	Formula	OCATA/ TA size	OK ?	Regions	Reduced regions	Zones	Reduced zones
2	$\Box \bigwedge_{i=1,2} ((o_i \wedge \Diamond_{[0,+\infty[} c_i) \Rightarrow \Diamond_{]1,2]} c_i)$	6 / 10	×	198 / 53	189 / 47	23 / 19	15 / 16
2	$\Box \bigwedge_{i=1,2} ((b_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,4]} o_i)$	6 / 10	✓	369 / 214	327 / 144	30 / 27	18 / 21
2	$\Box \bigwedge_{i=1,2} ((l_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,6]} o_i)$	6 / 10	✓	1,186 / 671	992 / 434	57 / 42	38 / 30
3	$\Box \bigwedge_{i=1,\ldots,3} ((o_i \wedge \Diamond_{[0,+\infty[} c_i) \Rightarrow \Diamond_{]1,2]} c_i)$	8 / 37	×	617 / 245	501 / 219	94 / 67	75 / 55
3	$\Box \bigwedge_{i=1,\ldots,3} ((b_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,12]} o_i)$	8 / 37	✓	>5min	>5min	646 / 365	526 / 218
3	$\Box \bigwedge_{i=1,\ldots,3} ((l_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,14]} o_i)$	8 / 37	✓	>5min	>5min	773 / 468	652 / 288
4	$\Box \bigwedge_{i=1,\ldots,4} ((o_i \wedge \Diamond_{[0,+\infty[} c_i) \Rightarrow \Diamond_{]1,2]} c_i)$	10 / 114	×	3,598 / 691	3,302 / 660	347 / 183	277 / 161
4	$\Box \bigwedge_{i=1,\ldots,4} ((b_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,20]} o_i)$	10 / 114	✓	>5min	>5min	6,411 / 2,665	3,135 / 1,458
4	$\Box \bigwedge_{i=1,\ldots,4} ((l_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,22]} o_i)$	10 / 114	✓	>5min	>5min	8,832 / 3,242	4,551 / 1,809
5	$\Box \bigwedge_{i=1,\ldots,5} ((o_i \wedge \Diamond_{[0,+\infty[} c_i) \Rightarrow \Diamond_{]1,2]} c_i)$	12 / 311	×	27,464 / 1,732	22,638 / 1,629	893 / 419	596 / 383
5	$\Box \bigwedge_{i=1,\ldots,5} ((b_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,28]} o_i)$	12 / 311	✓	>5min	OOM	164,008 / 13,128	71,807 / 7,056
5	$\Box \bigwedge_{i=1,\ldots,5} ((l_i \wedge \Diamond_{[0,+\infty[} o_i) \Rightarrow \Diamond_{[0,30]} o_i)$	12 / 311	✓	>5min	OOM	240,926 / 15,432	107,063 / 8,368

Finally, Tables 6.5 and 6.6 are dedicated to *infinite words*. Table 6.5 reports on satisfiability results, similarly to those of Table 6.1, except that we are now considering infinite words. Those results are of course obtained without using the order $\sqsubseteq$, which does not apply to the setting of infinite words. Table 6.6 displays model-checking results, similar to those presented in Table 6.3, but for the setting of infinite words. For this setting, we were able to produce satisfiable MITL formulas simpler than those established over finite words. Once again, the results of Table 6.6 are obtained without using any order.

Let us first consider the results obtained in Table 6.5. We start comparing results obtained on regions and reduced regions. In average, when considering the formulas containing an interval $[0, +\infty[$, the number of explored regions is divided by 2, and the running times are divided by 5, when using the reduced regions instead of the regions. On the other formulas, we have similar numbers of explored regions and the running time are rather variable: sometimes the running time is bigger when considering reduced regions, sometimes it is the opposite. We notice that when using reduced regions, our program gives an answer to 3 more tests than when using regions (by the time out of 5 minutes). Let us now consider the zones and the reduced zones. The numbers of explored zones are rather similar and the running times are in average higher by 10%, on *all* formulas, when using reduced zones instead of zones. We notice the exception of formula $R(5, [0, +\infty[)$ which gives an answer 30 times more quickly and explores 16 times less zones with the use of reduced zones.
We now compare the results obtained with regions and with zones. In general, the running times and the numbers of explored regions/zones are both divided by 3 when using zones instead of regions. We highlight the example of formula $E(10, [5, 8[)$ for which the running time is divided by 35 when using zones instead of regions. We also notice that the use of zones enables to answer 3 supplementary tests than that of regions (by 5 minutes): the 3 same problems that the reduced regions also enabled to solve.

We now consider the results of Table 6.6. Let us first compare results on

regions and reduced regions. In average, the use of reduced regions instead of regions only enables to reduce the number of explored regions by 3%, and the executions times are higher by 20%. Once again, these bigger running times are probably due to the supplementary computations necessary in the program to maintain reduced regions. We now observe results over zones and reduced zones. In general, the execution times are reduced by 10% when using reduced zones instead of zones, but on some examples, the execution time is higher by 10% as well. For the simplest formulas (i.e. when considering a lift with 2 floors), the number of explored zones are similar when using zones and reduced zones. The gap gradually widens to reach a running time reduced by 40% using reduced zones instead of zones, for formulas over a lift with 4 floors.

Let us finally compare results obtained on regions with that obatined on zones. We first notice that the use of zones enabled to answer 4 more problems than that of regions (by 5 minutes). On the tests on which both regions and zones answer, when using zones instead of regions, the running times are in average divided by 5 and the number of explored regions/zones is in general divided by 3, although it is divided by 13 on the example of formula $\Box \bigwedge_{i=1,2} \left(l_i \Rightarrow \Diamond_{[0,6]} o_i\right)$ for a lift with 2 floors. Surprisingly, the running time is bigger (by 10%) using zones on the formula $\Box \bigwedge_{i=1,\ldots,3} \left(o_i \Rightarrow \Diamond_{]1,2]} c_i\right)$ for a lift with 3 floors.

Table 6.5: Infinite words without order - Benchmark for the satisfiability. Reported values are execution time in ms / number of explored regions or zones.

Sat ?	Formula	Size	Regions	Reduced regions	Zones	Reduced zones
Sat	$E(5, [0, +\infty[)$	6	74 / 61	16 / 31	58 / 36	39 / 31
Sat	$E(10, [0, +\infty[)$	11	3,296 / 2,045	369 / 1,023	1,374 / 1,033	2,515 / 1,023
Sat	$E(5, [5, 8[)$	6	382 / 228	394 / 228	83 / 33	86 / 33
Sat	$E(10, [5, 8[)$	11	70,129 / 7,172	79,889 / 7,172	1,982 / 1,025	2,490 / 1,025
Unsat	$A(10, [0, +\infty[)$	11	1 / 1	1 / 1	4 / 1	5 / 1
Sat	$A(10, [5, 8[)$	11	1,926 / 7	2,036 / 5	3,036 / 2	3,153 / 2
Sat	$U(10, [0, +\infty[)$	10	231 / 7	5 / 4	16 / 1	6 / 1
Unsat	$U(2, [5, 8[)$	2	13 / 6	15 / 8	4 / 2	4 / 2
Unsat	$U(3, [5, 8[)$	3	OOM	OOM	OOM	OOM
Sat	$T(10, [0, +\infty[)$	10	$>$ 5min	3 / 2	33 / 3	7 / 2
Sat	$T(10, [5, 8[)$	10	52 / 2	40 / 2	11 / 2	11 / 2
Sat	$R(5, [0, +\infty[)$	21	$>$ 5min	301 / 270	4,307 / 1,321	145 / 81
Sat	$R(10, [0, +\infty[)$	41	$>$ 5min	OOM	OOM	$>$ 5min
Sat	$R(5, [5, 8[)$	21	OOM	6,996 / 117	1,299 / 36	1,518 / 36
Sat	$R(10, [5, 8[)$	41	$>$ 5min	$>$ 5min	$>$ 5min	$>$ 5min
Sat	$Q(5, [0, +\infty[)$	11	44 / 39	11 / 20	43 / 29	22 / 20
Sat	$Q(10, [0, +\infty[)$	21	1,209 / 1,041	286 / 521	841 / 540	933 / 521
Sat	$Q(5, [5, 8[)$	11	497 / 98	378 / 57	167 / 32	181 / 32
Sat	$Q(10, [5, 8[)$	21	35,776 / 2,646	20,324 / 2,912	81,774 / 782	86,228 / 782

Table 6.6: Infinite words without order - Benchmark for the model-checking. Reported values are execution time in ms / number of explored regions or zones.

Floors	Formula	OCATA/ TA size	OK ?	Regions	Reduced Regions	Zones	Reduced zones
2	$\Box \bigwedge_{i=1,2} (o_i \Rightarrow \Diamond_{]1,2]} c_i)$	4 / 10	×	132 / 90	124 / 87	57 / 35	56 / 32
2	$\Box \bigwedge_{i=1,2} (b_i \Rightarrow \Diamond_{[0,4]} o_i)$	4 / 10	✓	207 / 224	202 / 215	31 / 31	26 / 25
2	$\Box \bigwedge_{i=1,2} (l_i \Rightarrow \Diamond_{[0,6]} o_i)$	4 / 10	✓	554 / 690	769 / 669	68 / 51	60 / 40
3	$\Box \bigwedge_{i=1,\ldots,3} (o_i \Rightarrow \Diamond_{]1,2]} c_i)$	5 / 37	×	417 / 496	421 / 476	463 / 154	337 / 140
3	$\Box \bigwedge_{i=1,\ldots,3} (b_i \Rightarrow \Diamond_{[0,12]} o_i)$	5 / 37	✓	>5min	>5min	1,148 / 541	1,008 / 376
3	$\Box \bigwedge_{i=1,\ldots,3} (l_i \Rightarrow \Diamond_{[0,14]} o_i)$	5 / 37	✓	>5min	>5min	1,321 / 774	1,387 / 540
4	$\Box \bigwedge_{i=1,\ldots,4} (o_i \Rightarrow \Diamond_{]1,2]} c_i)$	6 / 114	×	2,228 / 1,697	2,813 / 1,639	1,381 / 498	1,565 / 461
4	$\Box \bigwedge_{i=1,\ldots,4} (b_i \Rightarrow \Diamond_{[0,20]} o_i)$	6 / 114	✓	>5min	>5min	26,146 / 5,757	22,776 / 3,156
4	$\Box \bigwedge_{i=1,\ldots,4} (l_i \Rightarrow \Diamond_{[0,22]} o_i)$	6 / 114	✓	>5min	>5min	52,167 / 7,577	48,754 / 4,337
5	$\Box \bigwedge_{i=1,\ldots,5} (o_i \Rightarrow \Diamond_{]1,2]} c_i)$	7 / 311	×	19,106 / 4,866	21,833 / 4,714	3,216 / 1,402	3,838 / 1,310
5	$\Box \bigwedge_{i=1,\ldots,5} (b_i \Rightarrow \Diamond_{[0,28]} o_i)$	7 / 311	✓	>5min	OOM	>5min	OOM
5	$\Box \bigwedge_{i=1,\ldots,5} (l_i \Rightarrow \Diamond_{[0,30]} o_i)$	7 / 311	✓	OOM	OOM	>5min	>5min

Part II

MITL Reactive Synthesis

CHAPTER 7

MITL BRSPlant algorithm

In this chapter, we consider the MITL reactive synthesis over finite words, with the pointwise semantics. In [17], Bouyer and al. proved that the MTL *bounded reactive synthesis with plant* (BRSPlant) is decidable. Unfortunately, they proposed an algorithm based on OCATA and well quasi ordering, which only gives a non primitive recursive bound. The aim of this chapter is to provide an algorithm to solve the MITL BRSPlant problem with a better upper bound. We present an algorithm following the outlines of that presented in [17].
This algorithm first constructs a timed game: this is the object of Section 7.1. In Section 7.2, we present how the resolution of this timed game can be turned into the analysis of a transition system of a particular kind: a *symbolic* transition system. Section 7.3 then displays the algorithm, using the previously constructed symbolic transition system, enabling to solve the MITL BRSPlant problem.
In these sections, we will so exhibit the objects progressively considered in [17], with the aim to solve the MTL BRSPlant problem. Nevertheless, our adaptation is twofold. On the one hand, we consider a version of the MITL BRSPlant problem with a new kind of concurrency, which was not present in the definitions

of [17]. Indeed, we only allow the environment to play a timed action if its delay is smaller than that the controller would like to play. This type of concurrency (also used in [28]) between the controller and the environment seems useful to enable to intuitively construct plants for the BRSPlant problem. On the second hand, we use OCATA interpreted over our interval semantics instead of the classical one: this trick enables to bound the number of clock copies used and so, to reduce the complexity of the algorithm of [17].
We end this chapter by proving, in Section 7.4, that to use our interval semantics enables to improve the upper bound of the algorithm on MTL for the particular case of MITL. Indeed, the algorithm presented in [17] for MTL has a non primitive recursive upper bound while our algorithm for MITL executes in a time triply exponential in the sizes of the MITL formula and the plant.

For the rest of this chapter, we are mainly considering the BRSPlant problem *for MITL*. Note that MITL formulas only use constants in $\mathbb{N}$, so that, in the sequel, we will only use granularities of type $\mu = (X, 1, K)$. For the sake of simplicity, we consider all along this chapter that a granularity is simply a pair $\mu = (X, K)$ where X is a finite set of clocks, and $K \in \mathbb{N}$.

7.1 Towards a timed game

In this section, we describe the first step of the procedure of [17] to solve the MITL BRSPlant problem. It consists in reducing it to a timed game between the environment and the controller.

We first need to fix some necessary vocabulary about guards, granularities and STSs.

Definition 7.1. *Let $\mu = (X, c_{max})$ be a granularity, we define the size of μ to be:*

$$|\mu| = \begin{cases} |X| + log_2(c_{max}) & \text{if } c_{max} \neq 0 \\ |X| & \text{otherwise} \end{cases}$$

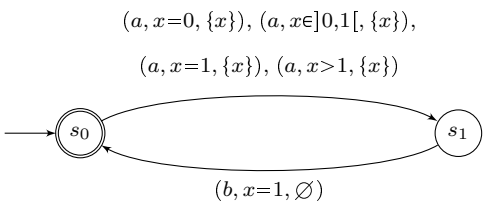


Figure 7.1: An atomic STS.

Definition 7.2. *Let X be a set of clocks and $g \in \mathcal{G}(X)$. We note $[\![g]\!]$ the set of valuations over X which satisfy the guard g.*

Definition 7.3. *The granularity of an STS $\mathcal{ST}$ is $\mu = (X, c_{max})$ if X is the set of clocks occuring in the constraints of $\mathcal{ST}$ and c_{max} is the largest constant occuring in the constraints of $\mathcal{ST}$.*

Definition 7.4. *Let $\mu = (X, K)$ be a granularity. A μ-granular constraint g is μ-atomic if for every μ-granular constraint g', either $[\![g]\!] \subseteq [\![g']\!]$ or $[\![g]\!] \cap [\![g']\!] = \varnothing$.*

Definition 7.5. *We say that an STS $\mathcal{ST}$ of granularity μ is* atomic *if each clock constraint of $\mathcal{ST}$ is μ-atomic.*

Example 7.6. Let us consider the granularity $\mu = (\{x\}, 8)$, we have : $|\mu| = 1 + log_2(8) = 4$.
Let us note g the guard $x > 2 \wedge x \leqslant 3$. Then, $[\![g]\!] = \{v : \{x\} \mapsto \mathbb{R}^+ \mid v(x) \in]2, 3]\}$.
Let us note g' the guard $x > 2 \wedge x < 3$. g and g' are μ-granular constraints (see definition 2.147). g is not μ-atomic because $[\![g]\!] \cap [\![g']\!] \neq \varnothing$ and $[\![g]\!] \nsubseteq [\![g']\!]$. However, g' is μ-atomic.
Let us consider the STS $\mathcal{ST}$ of Figure 7.1. Its granularity is $\nu = (\{x\}, 1)$ and it is atomic.

We are now able to recall the notion of *timed game* introduced in [31] and also used in [17].

Definition 7.7. *A timed game over finite words is a pair $G = (\mathcal{ST}, L)$, where $\mathcal{ST}$ is a symbol-deterministic STS with finitely many states, over a symbolic alphabet Γ based on (Σ, X) and $L \subseteq T\Sigma^\star$. Moreover, we require that $\mathcal{ST}$ is atomic.*

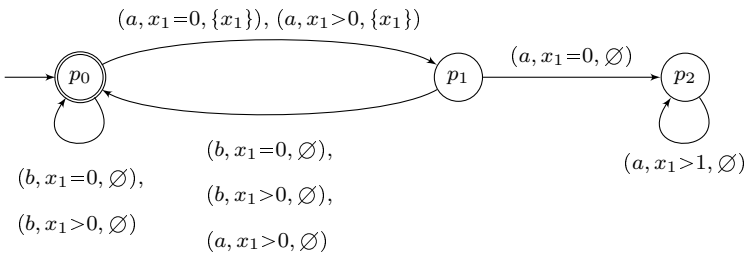


Figure 7.2: The atomic STS $\mathcal{ST}$.

In the rest of the chapter, we will be particularly interested in *MITL timed games*, which are timed games of the form $G = (\mathcal{ST}, L)$ such that L is the finite words language of an MITL formula, i.e. there is an MITL formula Φ such that $L = [\![\Phi]\!]$.

Example 7.8. Let us consider the atomic STS $\mathcal{ST}$ of Figure 7.2 and the following MITL formula:

$$\Phi \equiv \Box(a \Rightarrow \Diamond_{<1} b).$$

The pair $(\mathcal{ST}, [\![\Phi]\!])$ forms the timed game G.

Such a timed game is played between the controller and the environment as follows. The controller chooses a *valid* subset of symbolic actions (as defined below) enabled at the initial location of $\mathcal{ST}$ and the environment responds by choosing an action (σ_0, g_0, R_0) from that subset. Then, the controller chooses a valid subset of symbolic actions enabled at the resulting location and the environment picks (σ_1, g_1, R_1) from it, and so on. In the formal definition hereunder, we will see a choice of the controller and the corresponding response of the environment as a unique step of the game.

Let us come back to the notion of *validity*, formalized by a *validity function*. Intuitively, from the set U containing all the actions enabled at a given step of the game, the validity function gives the set of subsets of U that are *valid*. To be valid means that the controller cannot interfere with the clocks of the

environment. The controller is however allowed to reset the subset of its clocks it wants to reset. Finally, when a valid subset of U contains a symbolic action composed of an action of the controller and a guard g, it can only contain actions of the environment with a guard g' which is a 'time predecessor' of g, i.e. a choice of delay of the controller prevents the environment from playing later (but the environment can always overtake the controller).

Definition 7.9. *Given a symbolic alphabet* Γ *based on* (Σ, X) *with* $\Sigma = \Sigma_C \cup \Sigma_E$ *and* $X = X_C \cup X_E$[1]*, the* validity function *over* Γ *is the function valid* $: 2^\Gamma \to 2^{(2^\Gamma)}$ *such that every set of actions* $U \in 2^\Gamma$ *(representing the actions enabled at a given step of the game) is mapped to the nonempty set of* subsets of U *containing all ('valid') subsets* V *of* U *such that:*

1. *for each symbolic action of the form* $(\sigma, g, R) \in U$ *with* $\sigma \in \Sigma_E$ *and* $R \subseteq X_E$*:*
   (a) ***either*** V *includes* exactly one *action of the form* $(\sigma, g, R \cup R')$ *for some* $R' \subseteq X_C$[2]*,*
   (b) ***or****, noting* $g + t$ *the guard* g *in which each constant* k *is replaced by* $k + t$*,* V *does not contain any* (σ, g, R') *but contains* $(\sigma', g', R') \in U$ *such that* $\sigma' \in \Sigma_C$ *and* $\forall t \in \mathbb{R}^+$*,* $g + t \wedge g'$ *is not satisfiable*[3]*;*
2. *for each symbolic action of the form* $(\sigma, g, R) \in U$ *with* $\sigma \in \Sigma_C$*,* V *includes* at most one *action of the form* $(\sigma, g, R \cup R')$ *for some* $R' \subseteq X_C$*.*

Example 7.10. Let us consider again the timed game $G = (\mathcal{ST}, [\![\Phi]\!])$ of Example 7.8. Its STS has as symbolic alphabet Γ, based on $(\Sigma, \{x_1\})$ with $\Sigma = \Sigma_C \cup \Sigma_E$, $\Sigma_C = \{b\}$ and $\Sigma_E = \{a\}$, and $X_E = \{x_1\}$ while $X_C = \varnothing$. Let us consider the set

[1] representing the clocks of the controller and that of the environment.

[2] Here is our unique modification from the definition of [31], the original condition was simply: 'for each symbolic action of the form $(\sigma, g, R) \in U$ with $\sigma \in \Sigma_E$, V includes *exactly one* action of the form (σ, g, R') for some R''. Our modification aims to correspond to our adaptation of the notion of controller (which adds a new kind of concurrency between timed actions proposed by the controller and the environment).

[3] intuitively, this condition means that g is not a 'time predecessor' of g'.

$U \in 2^{\Gamma}$ of symbolic actions that can be performed from location p_0:

$$U = \{(a,\, x_1 = 0,\, \{x_1\}),\ (a,\, x_1 > 0,\, \{x_1\}),\ (b,\, x_1 = 0,\, \varnothing),\ (b,\, x_1 > 0,\, \varnothing)\}.$$

$valid(U) = \{V_1, V_2, V_3\}$ with:

- $V_1 := \{(a,\, x_1 = 0,\, \{x_1\}),\ (a,\, x_1 > 0,\, \{x_1\})\}$
  (it is obtained using twice the point 1. (a) of Definition 7.9 and using its point 2. with no action of the form $(\sigma, g, R \cup R'))$,
- $V_2 := \{(a,\, x_1 = 0,\, \{x_1\}),\ (a,\, x_1 > 0,\, \{x_1\}),\ (b,\, x_1 > 0,\, \varnothing)\}$
  (it is also obtained using twice the point 1. (a) of Definition 7.9 but using its point 2. with 1 action of the form $(\sigma, g, R \cup R'))$, and
- $V_3 := \{(a,\, x_1 = 0,\, \{x_1\}),\ (b,\, x_1 = 0,\, \varnothing)\}$
  (it is obtained using the point 1. (a) of Definition 7.9 for the symbolic action $(a,\, x_1 = 0,\, \{x_1\})$, the point 1. (b) of Definition 7.9 for the symbolic action $(a,\, x_1 > 0,\, \{x_1\})$, and using its point 2. with 1 action of the form $(\sigma, g, R \cup R'))$.

Before to present what is a *strategy* in a timed game $G = (\mathcal{ST}, L)$, we make a remark about symbol-deterministic STSs like $\mathcal{ST}$.

Remark 7.11. We already noticed that, in a *deterministic STS*, every timed word can be read by at most one path. In the same way, in a *symbol-deterministic STS* $\mathcal{ST} = (S, s_0, \Delta, S_f)$, every symbolic word γ can be read by at most one path. More formally, there exists at most one path π whose trace is γ. In that case, we denote by $\Delta(s_0, \gamma)$ the last location of the path π (if it exists).

We still need to define what is an *enabled* symbolic action.

Definition 7.12. *Let $\mathcal{ST} = (S, s_0, \Delta, S_f)$ be an STS, over the symbolic alphabet Γ based on (Σ, X), and a symbolic word γ over Γ. We say that a symbolic action $(a, g, R) \in \Sigma \times \mathcal{G}(X) \times 2^X$ is* enabled *in $\mathcal{ST}$ after reading γ if there exists a*

transition $(\Delta(s_0, \gamma), (a, g, R), s')$ *in* Δ *such that* $tw(\gamma \cdot (a, g, R)) \neq \emptyset$.
We note $\boldsymbol{Enabled}^{symb}_{\mathcal{ST}}(\gamma)$ *the set of symbolic actions enabled in* $\mathcal{ST}$ *after reading* γ.

We are now able to define a *strategy* in a timed game $G = (\mathcal{ST}, L)$. As previously explained, this strategy will represent the combination of an action of the controller (i.e. the choice of a valid set of actions) and all the possible responses of the environment (i.e. all the possible choices of one of the actions of the set proposed by the controller).

Definition 7.13. *Let* $\mathcal{ST} = (S, s_0, \Delta, F)$ *be an STS over the symbolic alphabet* Γ *and valid be the validity function over* Γ. *A strategy in* $\mathcal{ST}$ *is a mapping* $f : D \subseteq L_{symb}(\mathcal{ST}) \to 2^{\Gamma}$ *such that:*

- $\varepsilon \in D$ *and*
- *for all* $\gamma \in D$ *and* $b \in f(\theta)$: $f(\gamma) \in valid(\boldsymbol{Enabled}^{symb}_{\mathcal{ST}}(\gamma))$ *and* $\gamma \cdot b \in D$.

In the same way, a strategy in the timed game $G = (\mathcal{ST}, L)$ *is a strategy in* $\mathcal{ST}$.

Example 7.14. Let us consider again the timed game $G = (\mathcal{ST}, [\![\Phi]\!])$ of Examples 7.8 and 7.10. Let us consider again the set:

$$U := \{(a,\, x_1 = 0,\, \{x_1\}),\ (a,\, x_1 > 0,\, \{x_1\}),\ (b,\, x_1 = 0,\, \emptyset),\ (b,\, x_1 > 0,\, \emptyset)\}.$$

It is exactly the set $\mathbf{Enabled}^{symb}_{\mathcal{ST}}(\varepsilon)$. Recall that $valid(U) = \{V_1, V_2, V_3\}$ (see Example 7.10). In particular, a strategy f in $\mathcal{ST}$ can only be a function associating either V_1, or V_2, or V_3 to ε.

Such a strategy f induces a particular *set of plays* in a timed game G: we here formally define this notion.

Definition 7.15. *Let* $G = (\mathcal{ST}, L)$ *be a timed game whose STS has* Γ *as symbolic alphabet. Let* f *be a strategy in* $\mathcal{ST}$. *The* set of plays of f, *denoted by* $plays(f)$, *is the set of symbolic words of* $L_{symb}(\mathcal{ST})$ *that are consistent with* f, *i.e.:*

$$\gamma \in plays(f) \quad iff \quad for\ every\ prefix\ of\ \gamma\ of\ the\ form\ \gamma' \cdot b\colon\ b \in f(\gamma').$$

We are now able to define when the controller wins such a timed game $G = (\mathcal{ST}, L)$.

Definition 7.16. *Let $G = (\mathcal{ST}, L)$ be a timed game whose STS $\mathcal{ST} = (S, s_0, \Delta, F)$ has Γ as symbolic alphabet. Let f be a strategy in $\mathcal{ST}$.*
We say that f is winning (for the controller) iff for every play $\gamma \in plays(f)$, we have:

$$tw(\gamma) \subseteq L.$$

Here is an interesting Proposition of paper [17] linking the BRSPlant problem[4] and this notion of timed game. Its proof stays correct despite our slight modification of the definition of *controller* thanks to our corresponding modification of the notion of *validity function*.

Proposition 7.17 ([17]). Given a plant $\mathcal{P}$ over a symbolic alphabet Γ and of set of clocks X, a granularity $\mu = (X \cup X_{\mathcal{C}}, c_{max})$ and a timed language L over finite words, one can construct a timed game $G = (\mathcal{ST}_{\mathcal{P}}, L)$ such that $\mathcal{ST}_{\mathcal{P}}$ has granularity μ and

there exists a deterministic STS $\mathcal{ST}$ over Γ such that $T\Sigma^{\star}_{\mathcal{ST},\mathcal{P}} \cap L(\mathcal{P}) \subseteq L \cup \{\varepsilon\}$
iff
there is a winning strategy in G.

Here is the formal definition of $\mathcal{ST}_{\mathcal{P}}$, obtained from a plant $\mathcal{P}$ (see [31]).

Definition 7.18. *Let us consider a plant $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$ over a symbolic alphabet Γ based on (Σ, X) and a granularity $\mu = (X \cup X_{\mathcal{C}}, c_{max})$. $\mathcal{ST}_{\mathcal{P},\mu}$ is the symbol-deterministic STS $(P, p_0, \delta', F^{\mathcal{P}})$ such that $(p, (a, g', R \cup R'), p') \in \delta'$ iff:*

[4] It is easy to see that our definition of the BRSPlant problem (see Definition 2.158) is only a rewriting of the definition given in [17], but the addition of concurrency between timed actions proposed by the controller and the environment.

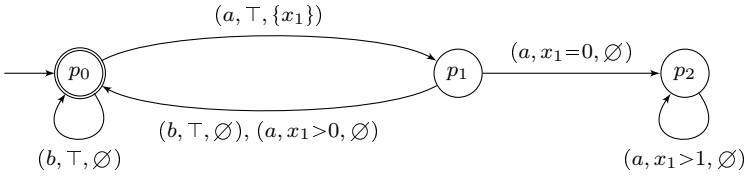


Figure 7.3: The atomic STS $\mathcal{ST}$.

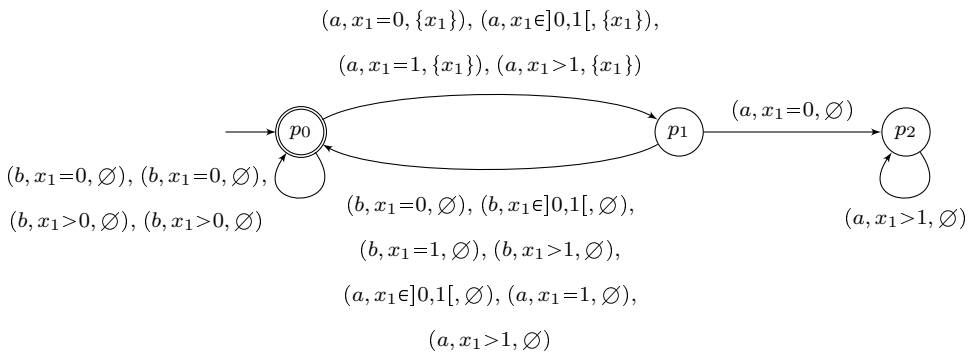


Figure 7.4: The STS $\mathcal{ST}_{\mathcal{P},\mu}$.

1. *$R' \subseteq X_{\mathcal{C}}$,*
2. *g' is μ-atomic, and*
3. *there exists $(p, (a, g, R), p') \in \delta$ such that $[\![g']\!] \subseteq [\![g]\!]$.*

Example 7.19. Let us consider the plant $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$ (see Figure 7.3) over a symbolic alphabet Γ based on $(\{a, b\}, \{x_1\})$, with $\Sigma = \Sigma_C \cup \Sigma_E$, $\Sigma_C = \{b\}$ and $\Sigma_E = \{a\}$.
First, let us observe the granularity $\mu = (\{x_1\}, 1)$, i.e. the controller has no proper clock. $\mathcal{ST}_{\mathcal{P},\mu}$ is the symbol-deterministic STS given in Figure 7.4.
If we now consider the granularity $\nu = (\{x_1, y\}, 0)$, i.e. the controller has one clock y. $\mathcal{ST}_{\mathcal{P},\nu}$ is the symbol-deterministic STS given in Figure 7.5.

Thanks to Proposition 7.17, the MITL BRSPlant problem can be reduced to deciding the existence of a winning strategy in an MITL timed game. The

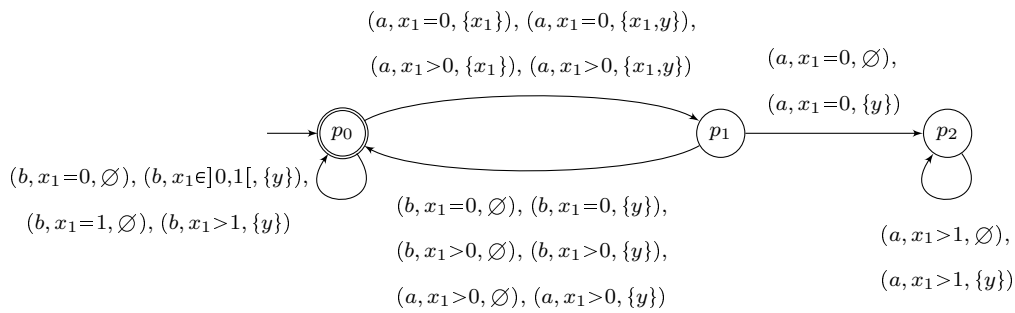


Figure 7.5: The STS $\mathcal{ST}_{\mathcal{P},\nu}$.

aim of the following section will be to reduce the existence of a winning strategy in an MITL timed game $G = (\mathcal{ST}_{\mathcal{P},\mu}, [\![\Phi]\!])$ to that of a winning strategy in a *symbol-deterministic STS*.

7.2 Towards a deterministic STS

In this section, we fix a plant $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$ over a symbolic alphabet Γ based on (Σ, X), an MITL formula Φ specifying desired behaviours and a granularity $\mu = (X \cup X_{\mathcal{C}}, c_{max})$. We consider the game $G_{\mathcal{P}} = (\mathcal{ST}_{\mathcal{P},\mu}, [\![\Phi]\!])$, where $\mathcal{ST}_{\mathcal{P},\mu} = (S, s_0, \Delta, F)$ and a *complete*[5] OCATA $\mathcal{A}_{\neg\Phi} = (\Sigma, L, \ell_0, F^{\Phi}, \delta)$ accepting $[\![\neg\Phi]\!]$ (see Definition 2.164), which will be interpreted over its $f^{\star}_{\Phi}$-semantics. Let us notice that it is not difficult to complete the OCATA given by Definition 2.164 for $\neg\Phi$ in way to obtain a *complete* OCATA accepting $[\![\neg\Phi]\!]$.
The aim of this section is to reduce the existence of a winning strategy in $G_{\mathcal{P}}$ to that of a winning strategy in a *symbol-deterministic STS* representing the parallel composition of $\mathcal{ST}_{\mathcal{P},\mu}$ and the OCATA $\mathcal{A}_{\neg\Phi}$. This symbol-deterministic STS is obtained in several steps from $G_{\mathcal{P}}$. In the first one, we define an STS

[5] An OCATA $\mathcal{A} = (\Sigma, L, \ell_0, F, \delta)$ is said *complete* iff $\forall \ell \in L$, $\forall \sigma \in \Sigma$, $\forall t \in \mathbb{R}^+$, there exists a transition starting from ℓ, labelled by σ and such that t satisfies its guard.

that simulates the parallel executions of $\mathcal{ST}_{\mathcal{P},\mu}$ and $\mathcal{A}_{\neg\Phi}$. Then, in a way similar to that followed to perform MITL model-checking, we use the regions of $\equiv$ (see Definition 4.21) to obtain an STS with finitely many locations over the symbolic alphabet Γ. As already observed, an STS with finitely many locations is nothing but a timed automaton. Hence, we are then in front of an 'untimed' automaton (over a *symbolic* alphabet): it may be translated into the symbol-deterministic STS we were looking for, using the classical determinization method explained in Definition 2.17.

We now present the first part of the construction of the symbol-deterministic STS we are looking for: we define an STS that executes $\mathcal{ST}_{\mathcal{P},\mu}$ and $\mathcal{A}_{\neg\Phi}$ in parallel. We begin fixing some necessary vocabulary.

Definition 7.20. *An $\mathcal{ST}_{\mathcal{P},\mu}/\mathcal{A}_{\neg\Phi}$-configuration is a pair $((s,v),C)$, where (s,v) is a configuration of $\mathcal{ST}_{\mathcal{P},\mu}$ and C is a configuration of $\mathcal{A}_{\neg\Phi}$ containing at most $M(\neg\Phi)$ clock copies (see Definition 4.10).*

Definition 7.21. *We call* single step of $\mathcal{ST}_{\mathcal{P},\mu} = (S, s_0, \Delta, F)$ *a transition*

$$(s,v) \xrightarrow{a,g,R\,,\,t} (s',v')$$

such that:

- (s,v) *and* (s',v') *are two configurations of* $\mathcal{ST}_{\mathcal{P},\mu}$,
- $(s,(a,g,R),s') \in \Delta$ *is a transition of* $\mathcal{ST}_{\mathcal{P},\mu}$,
- $v + t \in [\![g]\!]$, *and*
- $v' = (v+t)[R \leftarrow 0]$.

Definition 7.22. *For $(a,g,Y) \in \Gamma$, we define:*

- $Succ^{\mathcal{ST}_{\mathcal{P},\mu}}((s,v),t,(a,g,R)) := \{(s',v') \mid (s,v) \xrightarrow{a,g,R\,,\,t} (s',v')$ *is a single step of* $\mathcal{ST}_{\mathcal{P},\mu}\}$,

- $Succ^{\mathcal{A}_{\neg\Phi}}(C,t,\sigma) := \{C' \mid C \xrightarrow{t,\sigma}_{f^\star_{\neg\Phi}} C' \text{ in } \mathcal{A}_{\neg\Phi}\}$.

Thanks to those useful notations, we are now able to formally define the STS $\mathcal{ST}_{\mathcal{P},\neg\Phi}$ representing the parallel execution of $\mathcal{ST}_{\mathcal{P},\mu}$ and $\mathcal{A}_{\neg\Phi}$.

Definition 7.23. *Let $\mathcal{ST}_{\mathcal{P},\mu} = (S, s_0, \Delta, F)$ and $\mathcal{A}_{\neg\Phi} = (\Sigma, L, \ell_0, F, \delta)$. We define the STS $\mathcal{ST}_{\mathcal{P},\neg\Phi} = (U, u_0, \longrightarrow\!\!\!\!\rightarrow)$, where:*

- *U is the set of $\mathcal{ST}_{\mathcal{P},\mu}/\mathcal{A}_{\neg\Phi}$-configurations,*
- *$u_0 = ((s_0, v_0), \{(\ell_0, 0)\})$, where v_0 is the valuation such that $v_0(x) = 0$, $\forall x \in X$, corresponds to the initial $\mathcal{ST}_{\mathcal{P},\mu}/\mathcal{A}_{\neg\Phi}$-configuration, and*
- *$((s,v), C) \overset{a,g,R}{\longrightarrow\!\!\!\!\rightarrow} ((s',v'), C')$ iff $\exists t \in \mathbb{R}^+$ such that $C' \in Succ^{\mathcal{A}_{\neg\Phi}}(C,t,\sigma)$ and $(s',v') \in Succ^{\mathcal{ST}_{\mathcal{P},\mu}}((s,v),t,(a,g,R))$.*

Example 7.24. We consider again the plant $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$ of Figure 7.6, the granularity $\mu = (\{x_1\}, 1)$ and the symbol-deterministic STS $\mathcal{ST}_{\mathcal{P},\mu}$ (see Figure 7.6).
We consider the following MITL formula representing the desired behaviours of $\mathcal{P}$:

$$\Phi \equiv \Box\,(a \Rightarrow \Diamond_{<1} b)\,.$$

Figure 7.7 represents the completed OCATA $\mathcal{A}_{\neg\Phi}$ (remark that location $\neg\Phi_{init}$ of Definition 2.164 has been omitted for simplicity) recognizing $[\![\neg\Phi]\!]$.
Figure 7.8 gives a part of the STS $\mathcal{ST}_{\mathcal{P},\neg\Phi}$: each of its nodes has potentially an uncountable number of successors. We are not able to represent all the $\mathcal{ST}_{\mathcal{P},\mu}/\mathcal{A}_{\neg\Phi}$-configurations, so that we represent most of them by a unique *parametrized* node. For instance, we draw a unique node labelled $((p_1, 0), \{(\Diamond, t_1)\})$, with $t_1 \in]0,1[$, while in the real infinite STS $\mathcal{ST}_{\mathcal{P},\neg\Phi}$, there is one such node for each $t_1 \in]0,1[$ and as many arcs, all labelled by the same symbolic letter (a, $x_1 \in]0,1[$, $\{x_1\}$), leading to them. Moreover, when we develop the transitions from a node, we represent all the possible transitions labelled by a symbolic letter whose letter of Σ is a. We never represent the transitions on letter b (for the sake of readability).

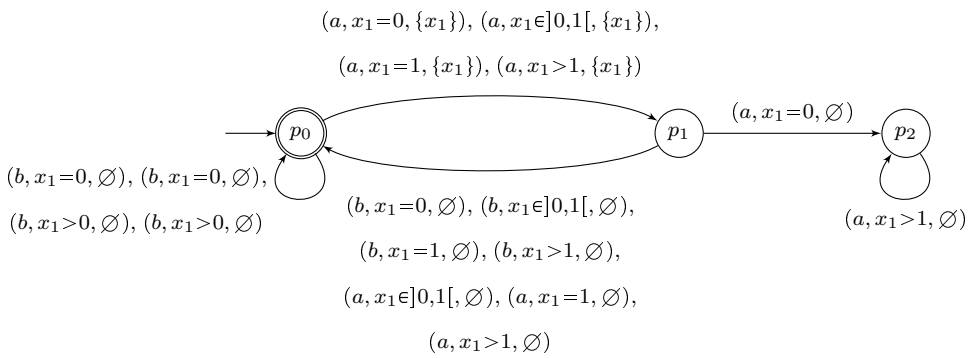


Figure 7.6: The STS $\mathcal{ST}_{\mathcal{P},\mu}$.

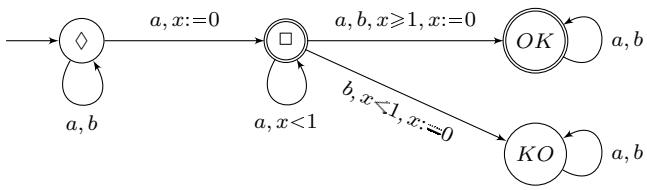


Figure 7.7: The complete OCATA $\mathcal{A}_{\neg\Phi}$.

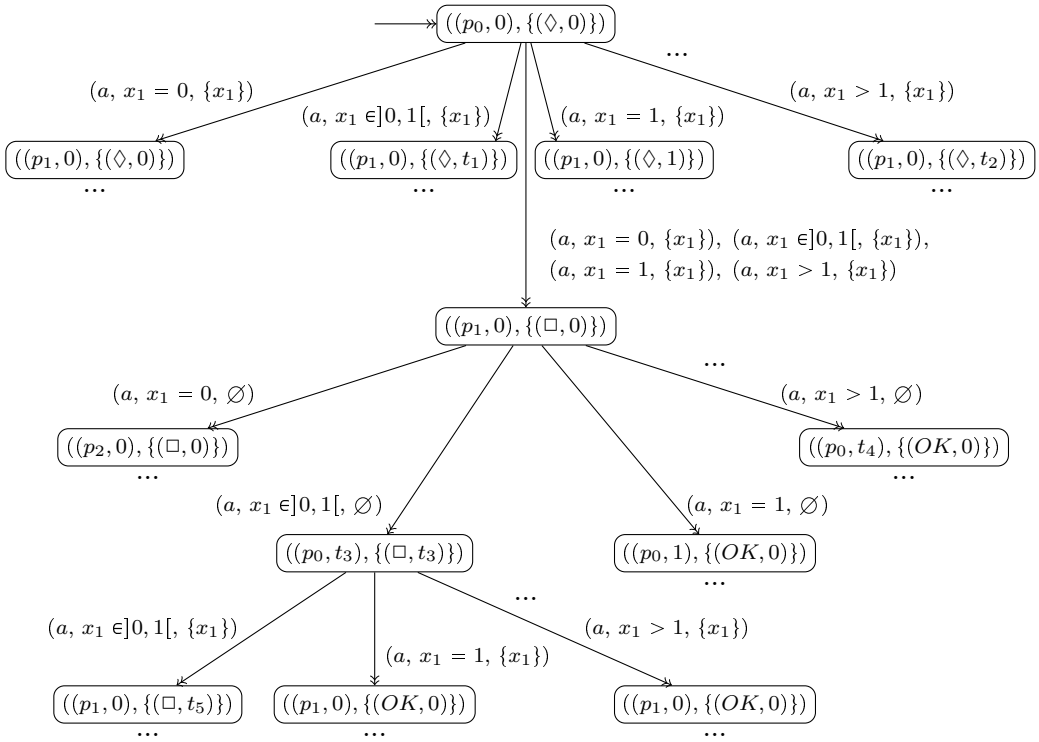
The following proposition holds by construction of $\mathcal{ST}_{\mathcal{P},\neg\Phi}$ and thanks to Theorem 4.7:

Proposition 7.25. For every plant $\mathcal{P}$, every MITL formula Φ, the associated complete OCATA $\mathcal{A}_{\neg\Phi}$, and the associated timed game $G_{\mathcal{P}} = (\mathcal{ST}_{\mathcal{P},\mu}, [\![\Phi]\!])$:

$$L_{symb}(\mathcal{ST}_{\mathcal{P},\neg\Phi}) = L_{symb}(\mathcal{ST}_{\mathcal{P},\mu} \times \mathcal{A}_{\neg\Phi}) \text{ and } L(\mathcal{ST}_{\mathcal{P},\neg\Phi}) = L(\mathcal{ST}_{\mathcal{P},\mu} \times \mathcal{A}_{\neg\Phi}).$$

Unfortunately, $\mathcal{ST}_{\mathcal{P},\neg\Phi}$ has infinitely many locations. However, as already noticed on a timed transition system representing the parallel execution of a TA and an OCATA, to solve the MITL model-checking problem, the relation $\equiv$ (see Definition 4.21) is a bisimulation:

Proposition 7.26. Let us consider $\mathcal{ST}_{\mathcal{P},\neg\Phi} = (U, u_0, \longrightarrow\!\!\!\!\rightarrow)$ (see Definition 7.23). Let $((s_1, v_1), C_1), ((s_2, v_2), C_2) \in U$ such that $((s_1, v_1), C_1) \equiv ((s_2, v_2), C_2)$. Then:



with: t_1 and $t_3 \in]0, 1[$; t_2, t_4 and $t_6 > 1$; $t_5 \in]0, 1[$ and $t_5 \geqslant t_3$.

Figure 7.8: Representation of a part of $\mathcal{ST}_{\mathcal{P},\neg\Phi}$.

for each transition $((s_1, v_1), C_1) \xrightarrow{a,g,R} ((s'_1, v'_1), C'_1)$, where (a, g, R) is a symbolic action and $((s'_1, v'_1), C'_1) \in U$, there exists $((s'_2, v'_2), C'_2) \in U$ such that:

$$((s_2, v_2), C_2) \xrightarrow{a,g,R} ((s'_2, v'_2), C'_2) \quad \text{and} \quad ((s'_1, v'_1), C'_1) \equiv ((s'_2, v'_2), C'_2).$$

The proof of this proposition is very close to that of Proposition 4.23.

We are so allowed to quotient the STS $\mathcal{ST}_{\mathcal{P},\neg\Phi}$ by the region bisimulation $\equiv$. Thanks to the bound $M(\neg\Phi)$ on the number of clock copies in configurations of $\mathcal{A}_{\neg\Phi}$, we obtain the STS *with finitely many locations* $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$ defined below. The regions of $\equiv$ are still represented by words of $\mathcal{H}$ (see Definition 4.25). We recall that the alphabet on which are written the words symbolically representing the region of each $s \in \mathcal{ST}_{\mathcal{P},\neg\Phi}$ is denoted Λ and consists in the subsets of:

$$(P \cup L) \times REG \times \{1, 2, \ldots, \max(\max_{\ell \in L}(\max_\ell), n)\},$$

where $\max_\ell$ is the maximal number of intervals that can be present in location $\ell \in L$ of $\mathcal{A}_{\neg\Phi}$ (given by the proof of Theorem 4.7), and $n := |X|$ the number of clocks of $\mathcal{P}$.
The function $H : S \to \Lambda^\star$ maps its region to each $s \in \mathcal{ST}_{\mathcal{P},\neg\Phi}$.

Definition 7.27. *Let us consider* $\mathcal{ST}_{\mathcal{P},\neg\Phi} = (U, u_0, \longrightarrow\!\!\!\!\!\rightarrow)$ *(see Definition 7.23).* $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi} = (W, w_0, \rightarrow)$ *is the STS such that:*

- $W = \{H(u) \mid u$ *is an* $\mathcal{ST}_{\mathcal{P}}/\mathcal{A}_{\neg\Phi}$*-configuration* $\}$,
- $w_0 = H(u_0)$,
- $w_1 \xrightarrow{a,g,R} w_2$ *iff there exists* $u_1 \in H^{-1}(w_1)$ *and* $u_2 \in H^{-1}(w_2)$ *such that* $u_1 \xrightarrow{a,g,R} u_2$.

We notice that the number of $\mathcal{ST}_{\mathcal{P}}/\mathcal{A}_{\neg\Phi}$-configurations is finite thanks to the existing bound on the number of clock copies of the $f^\star_{\neg\Phi}$-semantics of $\mathcal{A}_{\neg\Phi}$

given by Theorem 4.7. Hence, $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$ is a finite STS that can be effectively constructed.

Example 7.28. We consider again the STS $\mathcal{ST}_{\mathcal{P},\neg\Phi}$ given in Figure 7.8 (see Example 7.24). Figure 7.9 gives the part of the STS $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$ (consisting in words of $\mathcal{H}$) corresponding to the part of $\mathcal{ST}_{\mathcal{P},\neg\Phi}$ given in Figure 7.8. We recall that, in Figure 7.8, the node labelled $((p_1, 0), \{(\Diamond, t_1)\})$ (with $t_1 \in]0, 1[$) was a shorthand to represent the infinity of nodes really existing for each $t_1 \in]0, 1[$ and the infinity of arcs, all labelled by the same symbolic letter $(a,\ x_1 \in]0, 1[,\ \{x_1\})$, leading to each of them. In the present Figure 7.9 the node labelled by $\{(p_1, \{0\}, 1)\}\{(\Diamond,]0, 1[, 1)\}$ is not yet a shorthand but is really unique in $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$. Even though $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$ is finite, it has too many nodes to enable us to completely draw it.

We can now present the last step of the construction, whose aim is to obtain a symbol-deterministic STS from $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$. Remark that $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$ is an STS with finitely many locations, over a symbolic alphabet, so that $(\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi})_{det}$ (see Definition 2.17) is a finite symbol-deterministic STS, as we were looking for.
In the sequel, we however show that we can restrict the set of locations of $(\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi})_{det}$. We recall that a location of $(\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi})_{det}$ is a set of locations of $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$, i.e. a set of words of W. In fact, we claim we only need to consider the locations of $(\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi})_{det}$ which are sets of words agreeing on the configuration of $\mathcal{ST}_{\mathcal{P},\mu}$ they symbolically represent. This is due to the determinism of $\mathcal{P}$.

Definition 7.29. *For $w \in W$, we note $reg_{\mathcal{ST}_{\mathcal{P},\mu}}(w)$ the maximal subword $w' \sqsubseteq w$ such that w' does not contain occurrences of locations of $\mathcal{A}_{\neg\Phi}$.*

To prove our claim, let us further define a notion of *enabled symbolic action*, from a given word of W.

Definition 7.30. *Let us consider $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi} = (W, w_0, \rightarrow)$ (see Definition 7.27). We say that a symbolic action $(a, g, R) \in \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$ is* enabled *in $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$ at a word $w \in W$ if there exists $w' \in W$ such that $w \xrightarrow{a,g,R} w'$ in $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$. We note $\boldsymbol{Enabled}^{symb}_{\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}}(w)$ the set of symbolic actions enabled in $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$ at a word w.*

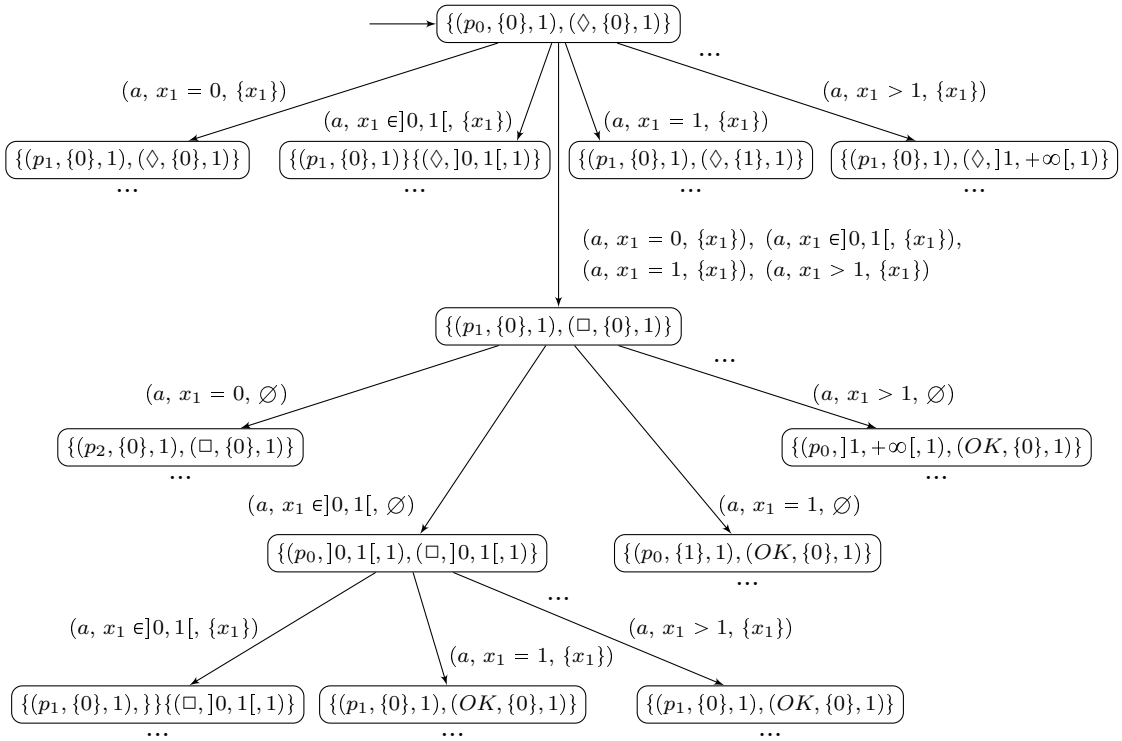


Figure 7.9: Representation of a part of $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$.

Since $\mathcal{A}_{\neg\Phi}$ is complete and $\mathcal{ST}_{\mathcal{P},\mu}$ is a symbol-deterministic atomic STS, we have:

Lemma 7.31. *For all $w_1, w_2, w_1', w_2' \in W$ with $reg_{\mathcal{ST}_{\mathcal{P},\mu}}(w_1) = reg_{\mathcal{ST}_{\mathcal{P},\mu}}(w_2)$: $w_1 \xrightarrow{a,g,R} w_1'$ and $w_2 \xrightarrow{a,g,R} w_2'$ imply that*

$$reg_{\mathcal{ST}_{\mathcal{P},\mu}}(w_1') = reg_{\mathcal{ST}_{\mathcal{P},\mu}}(w_2').$$

Moreover, $\boldsymbol{Enabled}^{symb}_{\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}}(w_1) = \boldsymbol{Enabled}^{symb}_{\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}}(w_2)$.

Thanks to this lemma, we are able to define the set of locations of $(\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi})_{det}$ we only need to consider. We then give the final definition of the symbol-deterministic STS $\mathcal{D}et$ we were looking for.

Definition 7.32. *We note SW the set of non-empty finite sets $\mathcal{D} \subseteq W$ such that for all words $w, w' \in \mathcal{D}$, $reg_{\mathcal{ST}_{\mathcal{P},\mu}}(w) = reg_{\mathcal{ST}_{\mathcal{P},\mu}}(w')$.*

Definition 7.33. *We note $\mathcal{D}et = (SW, \{w_0\}, \Delta_{\mathcal{D}et})$ the restriction of $(\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi})_{det}$ to the set of states of SW.*

The following lemma shows it is sufficient to consider $\mathcal{D}et$ instead of $(\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi})_{det}$ to answer language questions. It holds trivially, thanks to Lemma 7.31.

Lemma 7.34. $L_{symb}(\mathcal{D}et) = L_{symb}((\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi})_{det})$.

Let us extend the notion of *enabled symbolic action* to the locations of $\mathcal{D}et$.

Definition 7.35. *Let us consider $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi} = (W, w_0, \rightarrow)$ (see Definition 7.27) and $\mathcal{D}et = (SW, \{w_0\}, \Delta_{\mathcal{D}et})$ (see Definition 7.33). We say that a symbolic action $(a, g, R) \in \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$ is* enabled *in $\mathcal{D}et$ at a set of words $\mathcal{D} \in SW$ if there exists $w \in \mathcal{D}$, $w' \in W$ and a transition $w \xrightarrow{a,g,R} w'$ in $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$.*
We note $\boldsymbol{Enabled}^{symb}_{\mathcal{D}et}(\mathcal{D})$ the set of symbolic actions enabled in $\mathcal{D}et$ at a set of words $\mathcal{D}$.

We will now present a proposition showing that we can reduce the MITL BRSPlant problem (yet reduced to the search of a winning strategy in $G_{\mathcal{P}}$) to the search of a *safe strategy in* $\mathcal{D}et$, as defined below.

Definition 7.36. *An* $\mathcal{ST}_{\mathcal{P},\mu}/\mathcal{A}_{\neg\Phi}$*-configuration* $((s,v),C)$ *is bad if both* s *and* C *are accepting.*
A word $w \in W$ *is bad if there is* $((s,v),C) \in H^{-1}(w)$ *such that* $((s,v),C)$ *is bad.*
A set of words $\mathcal{D} \in SW$ *is bad if* $\mathcal{D}$ *contains some bad word.*

Definition 7.37. *A strategy* f *in* $\mathcal{D}et$ *is safe iff for every finite play* γ *of* f*,* $\Delta_{\mathcal{D}et}(\{w_0\},\gamma)$ *is* ***not*** *bad.*

Proposition 7.38 ([17])**.** There is a winning strategy in the timed game $G_{\mathcal{P}}$ with respect to undesired behaviours iff there is a safe strategy in $\mathcal{D}et$.

The proof of this proposition is identical to that presented in [17] despite the slight modifications we provided to the notions of *controller* and *validity function* in way to add concurrency between the controller and the environment of the BRSPlant problem.

Proposition 7.38 enables to conclude that the MITL BRSPlant problem can be reduced to deciding the existence of a safe strategy in the symbol-deterministic STS $\mathcal{D}et$.

Example 7.39. We consider again the STS $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$ given in Figure 7.9 (see

Example 7.24). We will use the following notations:

$$
\begin{aligned}
w_0 &:= \{(p_0, \{0\}, 1), (\Diamond, \{0\}, 1)\},\\
w_1 &:= \{(p_1, \{0\}, 1), (\Diamond, \{0\}, 1)\},\\
w_2 &:= \{(p_1, \{0\}, 1)\}\{(\Diamond,]0, 1[, 1)\},\\
w_3 &:= \{(p_1, \{0\}, 1), (\Diamond, \{1\}, 1)\},\\
w_4 &:= \{(p_1, \{0\}, 1), (\Diamond,]1, +\infty[, 1)\},\\
w_5 &:= \{(p_1, \{0\}, 1), (\Box, \{0\}, 1)\},\\
w_6 &:= \{(p_2, \{0\}, 1), (\Box, \{0\}, 1)\},\\
w_7 &:= \{(p_0,]0, 1[, 1), (\Box,]0, 1[, 1)\},\\
w_8 &:= \{(p_0, \{1\}, 1), (OK, \{0\}, 1)\},\\
w_9 &:= \{(p_0,]1, +\infty[, 1), (OK, \{0\}, 1)\},\\
w_{10} &:= \{(p_1, \{0\}, 1), \}\}\{(\Box,]0, 1[, 1)\},\\
w_{11} &:= \{(p_1, \{0\}, 1), (OK, \{0\}, 1)\},\\
w_{12} &:= \{(p_1, \{0\}, 1), (OK, \{0\}, 1)\},\\
w_{13} &:= \{(p_2, \{0\}, 1), (\Diamond, \{0\}, 1)\},\\
w_{14} &:= \{(p_2, \{0\}, 1), (\Box, \{0\}, 1)\},\\
w_{15} &:= \{(p_0,]0, 1[, 1), (\Diamond,]0, 1[, 1)\},\\
w_{16} &:= \{(\Box, \{0\}, 1)\}\{(p_0,]0, 1[, 1)\},\\
w_{17} &:= \{(p_0, \{1\}, 1), (\Diamond, \{1\}, 1)\},\\
w_{18} &:= \{(p_0, \{1\}, 1), (\Box, \{0\}, 1)\},\\
w_{19} &:= \{(p_0,]1, +\infty[, 1), (\Diamond,]1, +\infty[, 1)\},\\
w_{20} &:= \{(p_0,]1, +\infty[, 1), (\Box, \{0\}, 1)\}.
\end{aligned}
$$

$\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$ is then the STS partially represented in Figure 7.10. Figure 7.11 represents a part of $\mathcal{D}et$, whose bad locations are represented with double borders. It

comes from the fact that, in $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$, we have:

$$\begin{array}{l}
w_1 \xrightarrow{(a,x_1=0,\emptyset)} w_{13}, \\
w_1 \xrightarrow{(a,x_1=0,\emptyset)} w_{14}, \\
w_1 \xrightarrow{(a,x_1\in]0,1[,\emptyset)} w_{15}, \\
w_1 \xrightarrow{(a,x_1\in]0,1[,\emptyset)} w_{16}, \\
w_1 \xrightarrow{(a,x_1=1,\emptyset)} w_{17}, \\
w_1 \xrightarrow{(a,x_1=1,\emptyset)} w_{18}, \\
w_1 \xrightarrow{(a,x_1\in]1,+\infty[,\emptyset)} w_{19}, \\
w_1 \xrightarrow{(a,x_1\in]1,+\infty[,\emptyset)} w_{20}.
\end{array}$$

One can remark that Lemma 7.34 is indeed verified on this example. For instance, Let us observe the location $\{w_8, w_{17}, w_{18}\}$ of $\mathcal{D}et$: it contains three words agreeing on their trio '$(p_0,]1, +\infty[, 1)$', symbolically representing a configuration of $\mathcal{ST}_{\mathcal{P},\mu}$.

7.3 The algorithm

In the previous sections, we first showed that the MITL BRSPlant problem can be reduced to deciding the existence of a winning strategy in an MITL timed game (see Proposition 7.17). Then, we saw that this existence of winning strategy can itself be reduced to the existence of a safe strategy in a symbol-deterministic STS (see Proposition 7.38). In the present section, we present an algorithm verifying the existence of such a safe strategy.

Let us fix a plant $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$ over a symbolic alphabet Γ based on (Σ, X), an MITL formula Φ specifying desired behaviours and a granularity $\mu = (X \cup X_{\mathcal{C}}, c_{max})$. We consider the MITL BRSPlant problem determined by $\mathcal{P}$, Φ and μ. We also consider a complete OCATA $\mathcal{A}_{\neg\Phi} = (\Sigma, L, \ell_0, F^{\Phi}, \delta)$ accepting $[\![\neg\Phi]\!]$, the validity function *valid* and the symbol-deterministic STS $\mathcal{D}et = (SW, \{w_0\}, \Delta_{\mathcal{D}et})$ of Definition 7.33.

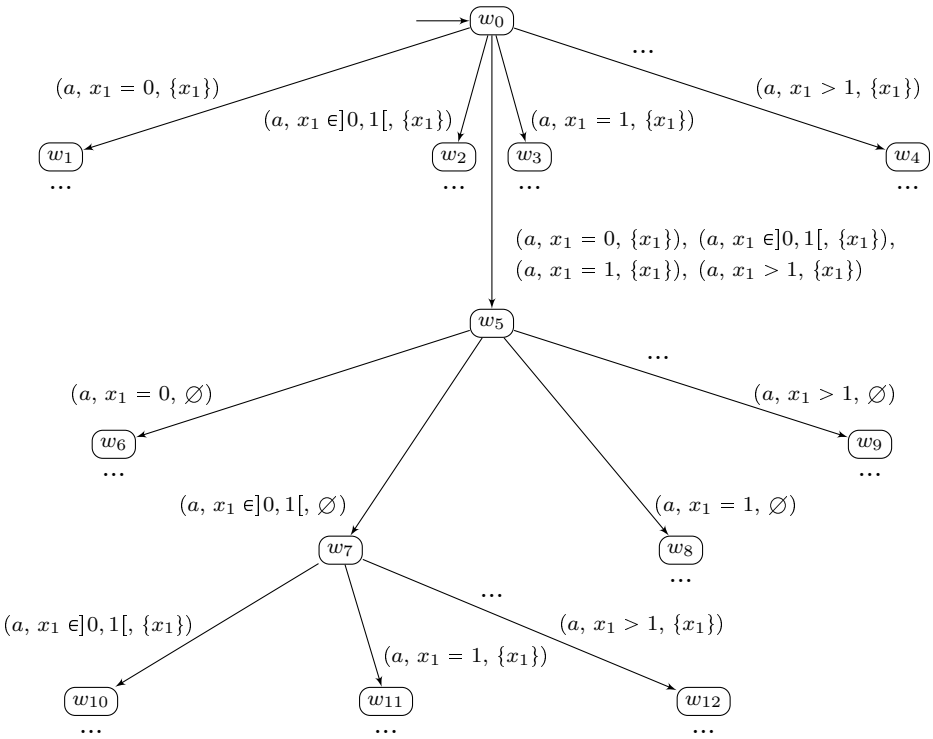


Figure 7.10: Representation of a part of $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$.

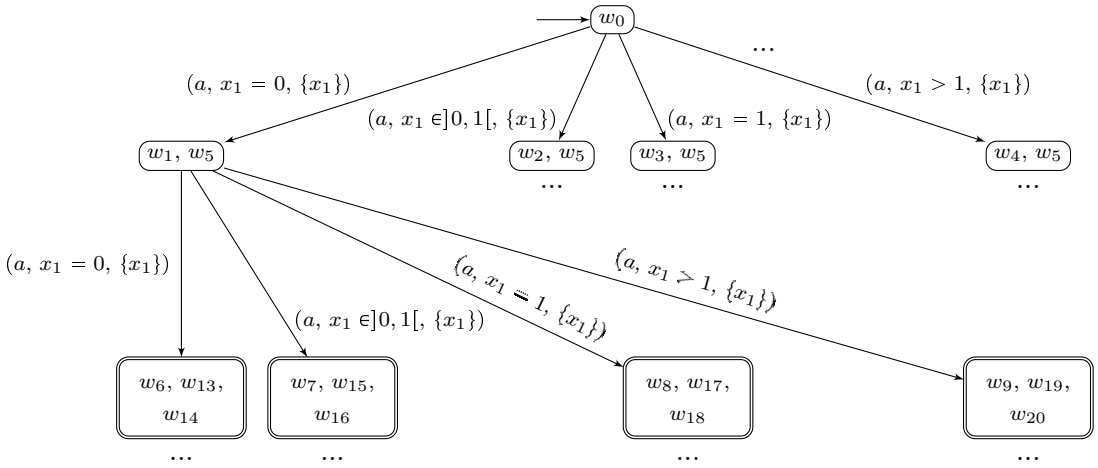


Figure 7.11: Representation of a part of $\mathcal{D}et$.

We now present an algorithm enabling to decide of the existence of a safe strategy in $\mathcal{D}et$. It is very close to that used in [17]. The unique difference is that, thanks to our bound on the number of clock copies in the configurations of $\mathcal{A}_{\neg\Phi}$, $\mathcal{D}et$ has finitely many locations. Hence, the following algorithm may stop without requiring the use of a well quasi order (as used in [17]).

Definition 7.40 (MITL BRSPlant algorithm)**.** *We build a finite portion $T^=$ of the tree given by the unfolding of $\mathcal{D}et$ and label its nodes by $\top$ or $\bot$. First of all, we construct $T^=$ and label its leaves. We start from the initial location $\{w_0\}$ and, at each step, we pick a leaf $\mathcal{D} \in SW$ and proceed as follows:*

1. *if $\mathcal{D}$ is not bad and it has an ancestor $\mathcal{D}'$ in the portion of the tree built so far such that $\mathcal{D}' = \mathcal{D}$, then we label the node $\mathcal{D}$ by $\top$ and do not compute its successors;*
2. *if $\mathcal{D}$ is bad, then we label the node $\mathcal{D}$ by $\bot$ and do not compute its successors;*
3. *otherwise, for any transition in $\mathcal{D}et$ of the form $(\mathcal{D}, (a, g, R), \mathcal{D}') \in \Delta_{\mathcal{D}et}$, we add a transition from the current node $\mathcal{D}$ to a new node $\mathcal{D}'$ labelled by (a, g, R). If $\mathcal{D}$ has no successor, then, it is labelled by $\top$.*

Remark that the construction of $T^=$ eventually ends because $\mathcal{D}et$ has only finitely many locations. Once $T^=$ is constructed, we label its internal nodes as follows. For any internal node $\mathcal{D}$, if there is a set of symbolic actions $U \in valid(\boldsymbol{Enabled}^{symb}_{\mathcal{D}et}(\mathcal{D}))$ such that for each $(a, g, R) \in U$, the transition in $T^=$ from $\mathcal{D}$ labelled by (a, g, R) leads to a node labelled by $\top$, then we label $\mathcal{D}$ by $\top$; otherwise, we label $\mathcal{D}$ by $\bot$. Finally, the algorithm answers 'yes' if the root of $T^=$ is labelled by $\top$ and 'no' otherwise.

The correctness of this algorithm is stated by the following proposition whose proof is identical to that given in [17].

Proposition 7.41 ([17])**.** If the algorithm of Definition 7.40 answers 'no', then there is no safe strategy in $\mathcal{D}et$. If this algorithm answers 'yes', then there is a safe strategy in $\mathcal{D}et$.

The following proposition gives an upper bound on the number of locations that may form the unfolding of the finite portion $T^{=}$ of the unfolding of $\mathcal{D}et$.

Proposition 7.42. The finite portion $T^{=}$ of the unfolding of $\mathcal{D}et$ constructed by the algorithm of Definition 7.40 has at most $O\left(a^{2^{2^b}}\right)$ locations, where:

- $a = |\Sigma|.(2.c_{max} + 2)^{|X|+|X_{\mathcal{C}}|}.2^{|X|+|X_{\mathcal{C}}|}$, and
- $b = (M(\Phi) + |X| + |X_{\mathcal{C}}|)\ .\ (|P| + |\Phi|)\ .\ (2.C_{max} + 2)\ .\ \max\big(|X| + |X_{\mathcal{C}}|, \max_{\ell\in L}(\max_\ell)\big)$, with $\max_\ell$ the maximal number of intervals that can be present in location $\ell \in L$ of $\mathcal{A}_{\neg\Phi}$ (given by the proof of Theorem 4.7).

Proof. We obtain this bound calculating step by step: the number of locations of $\mathcal{ST}_{\mathcal{P},\mu}$, the number of words of W (the set of locations of $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$), the number of set of words of SW (the set of locations of $\mathcal{D}et$) and finally, the number of locations of the unfolding of $\mathcal{D}et$ in which a branch is stopped as far as it reaches a location that has already been seen on this branch.
By definition of $\mathcal{ST}_{\mathcal{P},\mu}$ (see definition 7.18), $\mathcal{ST}_{\mathcal{P},\mu}$ has the same number of locations as $\mathcal{P}$, i.e. $|P|$ locations.
A location of $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$ is a word over the alphabet Λ consisting in the subsets of:

$$(P \cup L) \times REG \times \{1, 2, \ldots, \max\big(|X| + |X_{\mathcal{C}}|, \max_{\ell\in L}(\max_\ell)\big)\}.$$

which as between $|X|+|X_{\mathcal{C}}|$ and $M(\Phi)+|X|+|X_{\mathcal{C}}|$ letters. The maximal number of locations of $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$ is so:

$$\Sigma_{j=|X|+|X_{\mathcal{C}}|}^{M(\Phi)+|X|+|X_{\mathcal{C}}|}\quad 2^{j.m'} \quad = \quad O\left(2^{(M(\Phi)+|X|+|X_{\mathcal{C}}|).m'}\right),$$

where $m' = (|X| + |\Phi|).(2.c_{\max} + 2).\max\big(|X| + |X_{\mathcal{C}}|, \max_{\ell\in L}(\max_\ell)\big)$ (because $|L| = |\Phi|$).
A location of $\mathcal{D}et$ is a set of words of W, so that the number of locations of $\mathcal{D}et$ is

$$O\left(2^{2^{(M(\Phi)+|X|+|X_{\mathcal{C}}|).m'}}\right).$$

Finally, we want to calculate the number of locations of the unfolding of $\mathcal{D}et$ computed by the algorithm of Definition 7.40. In the worst case, all the locations of $\mathcal{D}et$ are present on each branch of this unfolding whose depth is so $O\left(2^{2^{(M(\Phi)+|X|+|X_{\mathcal{C}}|).m'}}\right)$. Moreover, from each of its locations, may be read all the symbolic actions of the symbolic alphabet $\Gamma' = \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$. There are

$$a \quad = \quad |\Sigma| \,.\, (2.c_{\max} + 2)^{|X|+|X_{\mathcal{C}}|} \,.\, 2^{|X|+|X_{\mathcal{C}}|}$$

such symbolic letters. Hence, the number of locations of the unfolding of $\mathcal{D}et$ is

$$O\left(a^{2^{2^{(M(\Phi)+|X|+|X_{\mathcal{C}}|).m'}}}\right) = O\left(a^{2^{2^b}}\right).$$

□

The following theorem is given to show that this upper bound on the number of locations we need to explore to solve the MITL BRSPlant problem corresponds to theoretical expectations. This observation is based on two ingredients. The first one consists in observing the complexity bound obtained by D'Souza and Madhusudan in [31] on a variant of the BRSPlant problem, in which the specification L represents *undesired behaviours* and is given by a *timed automaton*[6]. We remark that we can solve our own BRSPlant problem using that of [31]: we only need to negate the MITL formula Φ specificating desired behaviours and to construct the timed automaton $\mathcal{B}_{\neg\Phi}$ recognizing $[\![\neg\Phi]\!]$ (see Definition 4.13). This leads us to our second ingredient: the bound given by Proposition 4.15 on the number of locations of the timed automaton $\mathcal{B}_{\neg\Phi}$. Putting these ingredients together, we prove that the MITL BRSPlant problem is in 3EXPTIME.

Proposition 7.43. The MITL BRSPlant problem is in 3EXPTIME.

Proof. Let us consider the MITL BRSPlant problem given by the plant $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$, the MITL formula Φ and the granularity $\mu = (X, c_{\max})$. It is equivalent to construct from Φ the timed automaton $\mathcal{B}_{\neg\Phi}$ recognizing $[\![\neg\Phi]\!]$ and

[6] We however notice that, in [31], D'Souza and Madhusudan are considering infinite words.

then solve the variant of the BRSPlant problem of [31] for the specification of *undesired behaviours* given by the *timed automaton* $\mathcal{B}_{\neg\Phi} = (\Sigma, L, \lambda_0, X, F, \delta)$ (see Definition 4.13). Let us note $\nu = (X_\Phi, c_{max,\Phi})$ the granularity of $\mathcal{B}_{\neg\Phi}$. In [31], the authors proved that this problem can be solved in time:

$$|P|^{O(|L|)} \cdot 2^{O(|\Sigma|.|L|)+|L|^2.log(|L|).2^{O(|\mu|+|\nu|)^2}}.$$

We now use Proposition 4.15, which gives the number of clocks used by $\mathcal{B}_{\neg\Phi}$ and a bound on the number of locations of $\mathcal{B}_{\neg\Phi}$. Noting that $|\Phi| = |\neg\Phi|$, our MITL BRSPlant problem can be solved in time:

$$|P|^{O((|\Phi|)^{(m.|\Phi|)})} \cdot 2^{O(|\Sigma|.((|\Phi|)^{(m.|\Phi|)}))+((|\Phi|)^{(m.|\Phi|)})^2.log((|\Phi|)^{(m.|\Phi|)}).2^{O(|\mu|+|\nu|)^2}}$$
$$=|P|^{O(2^{m.|\Phi|.log_2(|\Phi|)})} \cdot 2^{O(|\Sigma|.(2^{m.|\Phi|.log_2(|\Phi|)}))+(2^{m.|\Phi|.log_2(|\Phi|)})^2.log(2^{m.|\Phi|.log_2(|\Phi|)}).2^{O(|X_\Phi|+log(c_{max,\Phi})+|\nu|)^2}},$$

where, noting $\mathcal{I}_\Phi$ the set of all the intervals that occur in Φ (and so in $\neg\Phi$), $m = \max_{I\in\mathcal{I}_\Phi}\left\{2 \times \left\lceil\frac{\inf(I)}{|I|}\right\rceil + 1, \left\lceil\frac{\sup(I)}{|I|}\right\rceil + 1\right\}$.
As $|X_\Phi| = M(\Phi) = O(2^{|\Phi|})$, the MITL BRSPlant problem is in 3EXPTIME. □

We close this section pointing out that we have tried to produce an efficient implementation of the algorithm of Definition 7.40. Indeed, the locations of the unfolding of $\mathcal{D}et$ are computed 'on the fly', as well as the words of W (i.e. locations of $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$) that form them. In fact, we start from the root of $\mathcal{D}et$, compute its successors and enumerate the *valid* subsets of successors, i.e. the subsets of successors reached following the arcs of each valid set of actions given by the validity function *valid*. We go on computing the successors of each such valid subset of successors, one by one. We stop the process as soon as a *bad* location of $\mathcal{D}et$ is reached, meaning that the valid set of succesors currently considered is not winning for the controller. If all the valid subsets of successors of a location of $\mathcal{D}et$ are eliminated this way, we label this location by $\perp$ and go back up in the unfolding of $\mathcal{D}et$. In the same way, when considering a location $\mathcal{D}$ of the unfolding of $\mathcal{D}et$ and a valid subset V of its successors, if all the locations of V can be labelled $\top$ (because they have no successor or had already been seen on the branch of the unfolding of $\mathcal{D}et$ they belong to), $\mathcal{D}$ is also labelled $\top$ and we go

back up in the unfolding of $\mathcal{D}et$. It means that, in this case, we do not compute the successors of the successors of $\mathcal{D}$ that did not belong to V.

7.4 Order-based algorithms

When we considered the MITL model-checking problem over finite words, we gave a basic algorithm and then tried to improve it elaborating a heuristic using *antichains*. In a similar way, we here consider *two* different orders enabling to elaborate such heuristics for the algorithm of Definition 7.40.
The first order has already been used by Bouyer and al. in [17]. Indeed, in this paper, they were considering the MTL BRSPlant problem, so that the unfolding of $\mathcal{D}et$ was an infinite object: they needed this order in way to stop its branches and ensure the termination of their algorithm. While, in our setting, the algorithm terminates without considering any order, we can use it, to improve it efficiency in practice. The first part of this section will be dedicated to the definition of the order used in [17]. This order considers two locations of the unfolding of $\mathcal{D}et$, i.e. set of words, and compare them. In parallel, we found interesting, as a location of the unfolding of $\mathcal{D}et$ is a set of words, to compare those words amongst themselves. This is the object of the second order we will consider in this section. In fact, we prove that we can simply keep the antichain of the minimal words (in the sense of Definition 4.38) among those forming each location of the unfolding of $\mathcal{D}et$, while keeping a correct algorithm for the MITL BRSPlant problem. Finally, we prove that these two orders may be used in a synchronous way, still keeping a correct algorithm.
In the following, we consider the MITL BRSPlant problem given by the plant $\mathcal{P} = (P, p_0, \delta, F^{\mathcal{P}})$, over a symbolic alphabet Γ based on (Σ, X), the MITL formula Φ specifying desired behaviours and the granularity $\mu = (X \cup X_{\mathcal{C}}, c_{max})$. We also consider a complete OCATA $\mathcal{A}_{\neg\Phi} = (\Sigma, L, \ell_0, F^{\Phi}, \delta)$ accepting $[\![\neg\Phi]\!]$, the validity function *valid* and the symbol-deterministic STS $\mathcal{D}et = (SW, \{w_0\}, \Delta_{\mathcal{D}et})$ of Definition 7.33.

First heuristic: stopping the branches earlier. Let us begin defining the order of Bouyer and al.. This order extends the definition of $\sqsubseteq$ (see Definition 4.38) to (**p**ower **s**et) locations of SW.

Definition 7.44. *We define the order $\sqsubseteq_{p.s.}$ on the locations of $\mathcal{D}et$ as follows. Let $\mathcal{D}_1, \mathcal{D}_2 \in SW$:*

$$\mathcal{D}_1 \sqsubseteq_{p.s.} \mathcal{D}_2 \quad \textit{iff} \quad \forall w_2 \in \mathcal{D}_2,\ \exists w_1 \in \mathcal{D}_1 \textit{ such that } w_1 \sqsubseteq w_2.$$

As $\sqsubseteq$ induces a forward simulation on the words of $\mathcal{H}$, $\sqsubseteq_{p.s.}$ induces a forward simulation on the sets of words of $\mathcal{D}et$:

Proposition 7.45 ([17])**.** For all $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}'_1 \in SW$ and $(a, g, R) \in \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$ such that

$$\mathcal{D}'_1 \sqsubseteq_{p.s.} \mathcal{D}_1 \text{ and } (\mathcal{D}_1, (a, g, R), \mathcal{D}_2) \in \Delta_{\mathcal{D}et},$$

there exists some $\mathcal{D}'_2 \in SW$ such that

$$\mathcal{D}'_2 \sqsubseteq_{p.s.} \mathcal{D}_2 \text{ and } (\mathcal{D}'_{,}(a, g, R), \mathcal{D}'_2) \in \Delta_{\mathcal{D}et}.$$

In [17], the algorithm of Definition 7.40 is proved to be correct when using the order $\sqsubseteq_{p.s.}$ instead of $=$ to truncate the branches of the unfolding of $\mathcal{D}et$. Let us formally define the finite part of this unfolding then constructed instead of $T^{=}$. We previously need to define the successor of a location of $\mathcal{D}et$ by a symbolic action (a, g, R).

Definition 7.46. *Let $\mathcal{D}$ be a location of $\mathcal{D}et = (SW, \{w_0\}, \Delta_{\mathcal{D}et})$ and $(a, g, R) \in \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$. We define $Post^{a,g,R}(\mathcal{D})$ to be the unique[7] $\mathcal{D}' \in SW$ such that $(\mathcal{D}, (a, g, R), \mathcal{D}') \in \Delta_{\mathcal{D}et}$ if it exists, and $\varnothing$ otherwise.*

Definition 7.47. *We define $T^{\sqsubseteq_{p.s.}}$ the finite part of the unfolding of $\mathcal{D}et = (SW, \{w_0\}, \Delta_{\mathcal{D}et})$ recursively obtained as follows:*

[7] by symbol-determinism of $\mathcal{D}et$

- *the root of* $T^{\sqsubseteq_{p.s.}}$ *is* $\{w_0\}$,
- *for each node* $\mathcal{D}$ *in* $T^{\sqsubseteq_{p.s.}}$, *if the branch of* $T^{\sqsubseteq_{p.s.}}$ *leading to* $\mathcal{D}$ *already contains another node* $\mathcal{D}'$, *such that* $\mathcal{D}' \sqsubseteq_{p.s.} \mathcal{D}$, *then,* $\mathcal{D}$ *has no successor; otherwise, for each symbolic action* $(a, g, R) \in \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$ *such that* $Post^{a,g,R}(\mathcal{D}) \neq \emptyset$, *there is an arc labelled by* (a, g, R) *from* $\mathcal{D}$ *to* $Post^{a,g,R}(\mathcal{D})$.

The algorithm of Definition 7.40 is still correct labelling the locations of $T^{\sqsubseteq}$ by $\top$ or $\bot$ in the same way as done with those of $T^{=}$ (see [17] for the correctness proof).

Second heuristic: minimal node labels. We now prove that it is sufficient to maintain the antichain of minimal words (for the order $\sqsubseteq$ of Definition 4.38) of the sets of words forming the locations of $\mathcal{D}et$, when developping its unfolding. In the sequel, we will note T the complete unfolding of $\mathcal{D}et$ and $T^{=}$ the finite part of T in which a branch is stopped when its last node is equal to another of its nodes. $T^{=}$ is the finite part of T used in the algorithm of Definition 7.40. We now define T_{min} and $T^{=}_{min}$, constructed similarly as T and $T^{=}$, but conserving $\min(\mathcal{D})$ instead of each location $\mathcal{D}$ of $\mathcal{D}et$.

Definition 7.48. *We define* T_{min} *the tree obtained as follows:*

- *the root of* T_{min} *is the same as that of* T,
- *for each node* $\mathcal{D}$ *of* T_{min}, *for each symbolic action* $(a, g, R) \in \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$ *such that* $Post^{a,g,R}(\mathcal{D}) \neq \emptyset$, *there is an arc labelled by* (a, g, R) *from* $\mathcal{D}$ *to* $\min(Post^{a,g,R}(\mathcal{D}))$.

Definition 7.49. *We define* $T^{=}_{min}$ *the tree obtained as follows:*

- *the root of* $T^{=}_{min}$ *is the same as that of* T,
- *for each node* $\mathcal{D}$ *of* $T^{=}_{min}$, *if the branch of* $T^{=}_{min}$ *leading to* $\mathcal{D}$ *already contains* $\mathcal{D}'$, *such that* $\mathcal{D}' = \mathcal{D}$, *then,* $\mathcal{D}$ *has no successor; otherwise, for each sym-*

bolic action $(a, g, R) \in \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$ *such that* $Post^{a,g,R}(\mathcal{D}) \neq \emptyset$, *there is an arc labelled by* (a, g, R) *from* $\mathcal{D}$ *to* $\min(Post^{a,g,R}(\mathcal{D}))$.

Our following aim will be to prove that labelling the nodes of $T^{=}_{min}$ instead of those of $T^{=}$ is correct in way to solve the MITL BRSPlant problem, as stated by the following theorem:

Theorem 7.50. *Using the procedure of the algorithm of Definition 7.40 to labelled the nodes of* $T^{=}$ *and* $T^{=}_{min}$, *we have that:*

the root of $T^{=}$ *is labelled by* $\top$ *iff the root of* $T^{=}_{min}$ *is labelled by* $\top$.

In way to prove this theorem, we will fix some vocabulary and expound several properties of $T^{=}$ and $T^{=}_{min}$.

Definition 7.51. *Let* $U = (V, E)$ *and* $U' = (V', E')$ *two directed trees whose respective roots are* r *and* r'. *We say that* U *and* U' *are* isomorphic *if there exists a bijection* $f : V \to V'$ *such that:* $f(r) = r'$ *and,* $\forall (v, w) \in E$, $(f(v), f(w)) \in E'$. *We call* corresponding nodes *nodes* $v \in V$ *and* $v' \in V'$ *such that* $f(v) = v'$.

Definition 7.52. *Let* $\mathcal{D}$ *be a node of* T, T_{min}, $T^{=}$ *or* $T^{=}_{min}$. *Let* $v \in valid($ $\boldsymbol{Enabled}^{symb}_{\mathcal{D}et}(\mathcal{D}))$ *be a valid set of actions among those enabled at* $\mathcal{D}$. *We say that* v *is* safe *if all the successors of* $\mathcal{D}$ *by a symbolic action from* v *are labelled by* $\top$ *by the algorithm of Definition 7.40.*

We now give several lemmas and propositions that will enable to establish the proof of Theorem 7.50. We start by stating two lemmas whose trivial proofs are omitted.

Lemma 7.53. *Let* $\mathcal{D}$ *be a location of* $\mathcal{D}et$. $\min(\mathcal{D}) \subseteq \mathcal{D}$ *and* $\min(\mathcal{D}) \sqsubseteq_{p.s.} \mathcal{D}$.

Lemma 7.54. *Let* $\mathcal{D}, \mathcal{D}'$ *be two locations of* $\mathcal{D}et$ *and* $(a, g, R) \in \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$. $\mathcal{D}' \subseteq \mathcal{D}$ *implies* $Post^{a,g,R}(\mathcal{D}') \subseteq Post^{a,g,R}(\mathcal{D})$[8].

[8] As previously noticed, $Post^{a,g,R}(\mathcal{D}')$ is a unique location of $\mathcal{D}et$, as well as $Post^{a,g,R}(\mathcal{D})$. The inclusion $Post^{a,g,R}(\mathcal{D}') \subseteq Post^{a,g,R}(\mathcal{D})$ concerns the words these two locations of $\mathcal{D}et$ contain.

Here is a useful proposition stating that, for any location $\mathcal{D}$ of $\mathcal{D}et$, the same symbolic actions are enabled from $\mathcal{D}$ and $\min(\mathcal{D})$.

Proposition 7.55. Let $\mathcal{D}$ be a location of $\mathcal{D}et = (SW, \{w_0\}, \Delta_{\mathcal{D}et})$ and $(a, g, R) \in \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$:

$$Post^{a,g,R}(\mathcal{D}) = \varnothing \quad \text{iff} \quad Post^{a,g,R}(\min(\mathcal{D})) = \varnothing.$$

Proof. ($\Rightarrow$) Trivial by definition of $\Delta_{\mathcal{D}et}$ (see also Definition 2.17) because $\min(\mathcal{D}) \subseteq \mathcal{D}$.
($\Leftarrow$) Suppose $Post^{a,g,R}(\min(\mathcal{D})) = \varnothing$. It means we have no successor reading (a, g, R) from the smallest words of $\mathcal{D}$ (for the order $\sqsubseteq$), by the definition of the transition relation $\rightarrow$ of $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$ and the definition of 'minimal model', we cannot have any successor reading (a, g, R) from any word of $\mathcal{D}$. So, $Post^{a,g,R}(\mathcal{D}) = \varnothing$. $\square$

The following proposition enables to prove that T and T_{min} are isomorphic. This fact will be a key argument in the proof of Theorem 7.50.

Proposition 7.56. Let $\mathcal{D}$ be a location of $\mathcal{D}et = (SW, \{w_0\}, \Delta_{\mathcal{D}et})$:

$$\min(Post^{a,g,R}(\mathcal{D})) = \min(Post^{a,g,R}(\min(\mathcal{D}))).$$

Proof. As $\mathcal{D}$ and $\min(\mathcal{D})$ are empty at the same time, this equality is trivially verified when one of them is empty. In the sequel, we suppose they are not empty.
($\subseteq$) Let $w \in \min(Post^{a,g,R}(\mathcal{D}))$. Let us note $\mathcal{D}' := Post^{a,g,R}(\mathcal{D})$. We have $(\mathcal{D}, (a, g, R), \mathcal{D}') \in \Delta_{\mathcal{D}et}$ and $w \in \min(\mathcal{D}')$. As $\min(\mathcal{D}) \sqsubseteq_{p.s.} \mathcal{D}$, by Proposition 7.45, noting $m\mathcal{D}' := Post^{a,g,R}(\min(\mathcal{D}))$, we have that $(\min(\mathcal{D}), (a, g, R), m\mathcal{D}') \in \Delta_{\mathcal{D}et}$ and $m\mathcal{D}' \sqsubseteq_{p.s.} \mathcal{D}'$. We must prove that $w \in \min(m\mathcal{D}')$.
<u>$w \in m\mathcal{D}'$:</u> as $w \in \min(\mathcal{D}')$, in particular, $w \in \mathcal{D}'$ and there is $\tilde{w} \in \mathcal{D}$ such that $\tilde{w} \xrightarrow{a,g,R} w$ (in $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$). We claim this implies there is also $w^\star \in \min(\mathcal{D})$ such

that $w^\star \xrightarrow{a,g,R} w$ (in $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$). Indeed, if $\tilde{w} \in \min(\mathcal{D})$, we are done. Otherwise, $\tilde{w} \notin \min(\mathcal{D})$ and there is $\overline{w} \in \min(\mathcal{D})$ such that $\overline{w} \sqsubseteq \tilde{w}$. As $\tilde{w} \xrightarrow{a,g,R} w$, by Proposition 4.43, there exists an mw such that $\overline{w} \xrightarrow{a,g,R} mw$ and $mw \sqsubseteq w$. As $\overline{w} \in \mathcal{D}$ and by definition of $Post^{a,g,R}(\mathcal{D})$, $mw \in \mathcal{D}'$. So, $w \in \min(\mathcal{D}')$ and $mw \sqsubseteq w$, which imply that $mw = w$: mw is the $w^\star$ we were looking for. Now, we know that there is $w^\star \in \min(\mathcal{D})$ such that $w^\star \xrightarrow{a,g,R} w$, and so $w \in m\mathcal{D}'$ (since $m\mathcal{D}' := Post^{a,g,R}(\min(\mathcal{D})))$.

$\underline{w \in \min(m\mathcal{D}')}$: $w \in \min(\mathcal{D}')$ means that $\forall w' \in \mathcal{D}'$, $w' \sqsubseteq w$ implies that $w' = w$. As $\min(\mathcal{D}) \subseteq \mathcal{D}$, by Lemma 7.54, $Post^{a,g,R}(\min(\mathcal{D})) := m\mathcal{D}' \subseteq \mathcal{D}' := Post^{a,g,R}(\mathcal{D})$. So, in particular, $\forall w' \in m\mathcal{D}'$, $w' \sqsubseteq w$ implies that $w' = w$.

($\supseteq$) Let $mmw \in \min(Post^{a,g,R}(\min(\mathcal{D})))$. Let us note $m\mathcal{D}' := Post^{a,g,R}(\min(\mathcal{D}))$. We have $(\min(\mathcal{D}), (a, g, R), m\mathcal{D}') \in \Delta_{\mathcal{D}et}$ and $mmw \in \min(m\mathcal{D}')$. As $\min(\mathcal{D}) \subseteq \mathcal{D}$ and $Post^{a,g,R}(\min(\mathcal{D}))$ is not empty, $Post^{a,g,R}(\mathcal{D})$ is not empty and we will note it $\mathcal{D}'$. We so have: $(\mathcal{D}, (a, g, R), \mathcal{D}') \in \Delta_{\mathcal{D}et}$ and $\min(\mathcal{D}) \sqsubseteq_{p.s.} \mathcal{D}$. By Proposition 7.45 and as $m\mathcal{D}'$ is the unique successor of $\min(\mathcal{D})$ reading (a, g, R) (by symbol-determinism of $\mathcal{D}et$), we have $(\min(\mathcal{D}), (a, g, R), m\mathcal{D}') \in \Delta_{\mathcal{D}et}$ and $m\mathcal{D}' \sqsubseteq_{p.s.} \mathcal{D}'$. We must show that $mmw \in \min(\mathcal{D}') := \min(Post^{a,g,R}(\mathcal{D}))$.

$\underline{mmw \in \mathcal{D}'}$: we know that $\min(\mathcal{D}) \subseteq \mathcal{D}$: by Lemma 7.54, $Post^{a,g,R}(\min(\mathcal{D})) := m\mathcal{D}' \subseteq \mathcal{D}' := Post^{a,g,R}(\mathcal{D})$. As $mmw \in \min(m\mathcal{D}')$, $mmw \in m\mathcal{D}'$ and so $mmw \in \mathcal{D}'$.

$\underline{mmw \in \min(\mathcal{D}')}$: $mmw \in \min(m\mathcal{D}')$ means that $\forall w' \in m\mathcal{D}'$, $w' \sqsubseteq mmw$ implies that $w' = mmw$. Let $x \in \mathcal{D}'$ and suppose $x \sqsubseteq mmw$. We will show that $x = mmw$. As $x \in \mathcal{D}'$ and $m\mathcal{D}' \sqsubseteq_{p.s.} \mathcal{D}'$, there is $mx \in m\mathcal{D}'$ such that $mx \sqsubseteq x$. As, moreover, $x \sqsubseteq mmw$, we have $mx \sqsubseteq mmw$ and so $mx = mmw$. Hence, $mmw \sqsubseteq x \sqsubseteq mmw$ and so $x = mmw$. □

Proposition 7.57. T and T_{min} are isomorphic. Moreover, for each node $\mathcal{D}$ in T, the corresponding node in T_{min} is labelled by $\min(\mathcal{D})$.

Proof. It is clear that T and T_{min} are isomorphic by definition of these trees and Proposition 7.56. We will prove that for each node $\mathcal{D}$ in T, the corresponding node in T_{min} is labelled by $\min(\mathcal{D})$.

Basis: the root $\{w_0\}$ of T is also the root of T_{min} and it contains a unique word, so that the root of T_{min} is indeed labelled by $\min(\{w_0\})$.
Inductive case: suppose that $\mathcal{D}$ and $min(\mathcal{D})$ are corresponding nodes of T and T_{min}, respectively. We must prove that for each symbolic action $(a, g, R) \in \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$, $\mathcal{D}$ is linked to a node $\mathcal{D}'$ by an arc labelled by (a, g, R) iff $min(\mathcal{D})$ is linked to a node $min(\mathcal{D}')$ by an arc labelled by (a, g, R). Indeed:

there is an arc labelled by (a, g, R) from $\mathcal{D}$ to a node $\mathcal{D}'$

iff	$\mathcal{D}' := Post^{a,g,R}(\mathcal{D})$	(by definition of T)
iff	there is an arc from $min(\mathcal{D})$, labelled by (a, g, R), to a node containing $\min(Post^{a,g,R}(\min(\mathcal{D})))$	(by definition of T_{min})
iff	there is an arc from $min(\mathcal{D})$, labelled by (a, g, R), to a node containing $\min(Post^{a,g,R}(\mathcal{D}))$	(by Proposition 7.56)
iff	there is an arc from $min(\mathcal{D})$, labelled by (a, g, R), to a node containing $\min(\mathcal{D}')$	(by definition of $\mathcal{D}$')

□

Nevertheless, $T^=$ and $T^=_{min}$ are *not* isomorphic. Indeed, each branch of $T^=_{min}$ is necessarily shorter than the corresponding branch in $T^=$ but the converse is not true.

Proposition 7.58. Each branch of $T^=_{min}$ is necessarily shorter than the corresponding branch in $T^=$.

Proof. If a branch of $T^=$ is stopped in a node $\mathcal{D}$ that has a corresponding node $min(\mathcal{D})$ in $T^=_{min}$, there are three possible reasons for this:

1. $\mathcal{D}$ has no successor. Then, $min(\mathcal{D})$ and has no successor, by Proposition 7.55.

2. there is $\mathcal{D}'$ on the branch leading to $\mathcal{D}$ such that $\mathcal{D}' = \mathcal{D}$. Then, on the branch leading to $min(\mathcal{D})$ is the node corresponding to $\mathcal{D}'$: $min(\mathcal{D}')$. As $\mathcal{D}' = \mathcal{D}$, $min(\mathcal{D}') = min(\mathcal{D})$ and $T^=_{min}$ is stopped in $min(\mathcal{D})$.

3. $\mathcal{D}$ is bad. Then, there is $w \in \mathcal{D}$ such that w is bad. As $\min(\mathcal{D}) \sqsubseteq_{p.s.} \mathcal{D}$ and $w \in H$, there is $mw \in \min(\mathcal{D})$ such that $mw \sqsubseteq w$. As w is bad (all the locations of $\mathcal{B}$ and $\mathcal{A}_{\neg\Phi}$ present in w are accepting ones), mw is bad and so is $\min(\mathcal{D})$. $T^{=}_{min}$ is so stopped in $min(\mathcal{D})$.

□

Remark 7.59. $T^{=}$ and $T^{=}_{min}$ are *not* isomorphic because the reverses of cases 2. and 3. are not true in general.

Remark 7.60. By definition of $T^{=}$ and $T^{=}_{min}$ and Proposition 7.55, each node of $T^{=}_{min}$ has: either no successor, either as many successors as its corresponding node in $T^{=}$.

We present a last lemma, used in the proof of Theorem 7.50. It links the valid set of actions existing from $\mathcal{D}$ and from $\min(\mathcal{D})$.

Lemma 7.61. *Let $\mathcal{D}$ be a node of T, $min(\mathcal{D})$ be its corresponding node in T_{min} and let $v \subseteq 2^{\Gamma}$. v is a valid set of actions from $\mathcal{D}$ iff v is a valid set of actions from $min(\mathcal{D})$, i.e.:*

$$v \in valid(\boldsymbol{Enabled}^{symb}_{\mathcal{D}et}(\mathcal{D})) \quad \textit{iff } v \in valid(\boldsymbol{Enabled}^{symb}_{\mathcal{D}et}(min(\mathcal{D})))$$

Proof. ($\Rightarrow$) Suppose that $v \in valid(\mathbf{Enabled}^{symb}_{\mathcal{D}et}(\mathcal{D}))$. Then, $v \in valid($ $\mathbf{Enabled}^{symb}_{\mathcal{D}et}(min(\mathcal{D})))$ because being valid from $\min(\mathcal{D})$ (see Definition 7.9) consists in satisfying properties on the actions readable from $\min(\mathcal{D})$, but v is valid from $\mathcal{D}$ and, by Proposition 7.55, the same labels are readable from $\min(\mathcal{D})$ and $\mathcal{D}$.
($\Leftarrow$) The same argument holds. □

Remark 7.62. The previous lemma also holds when considering a node $\mathcal{D}$ of $T^{=}$ that has a corresponding node in $T^{=}_{min}$ (in which case this corresponding node is necessarily $min(\mathcal{D})$).

Thanks to the previous lemmas and propositions, we are now ready to present the proof of Theorem 7.50. This shows that replacing $T^=$ by $T^=_{min}$ in the algorithm of Definition 7.40 does not change its correctness.

Proof of Theorem 7.50. ($\Rightarrow$) We will show the stronger following property:

if a node of $T^=$ that has a corresponding node in $T^=_{min}$ is labelled by $\top$, then its corresponding node is also labelled by $\top$.

Let $\mathcal{D}$ be a node of $T^=$ labelled by $\top$ by the algorithm of Definition 7.40, which has a corresponding node $min(\mathcal{D})$ in $T^=_{min}$. We will show that $min(\mathcal{D})$ is also labelled by $\top$ by this algorithm. There are three possible reasons for which $\mathcal{D}$ has been labelled by $\top$:

1. $\mathcal{D}$ has no successor. Then, $min(\mathcal{D})$ has no successor (see proposition 7.55): it is labelled by $\top$.

2. there is $\mathcal{D}'$ on the branch leading to $\mathcal{D}$ such that $\mathcal{D}' = \mathcal{D}$. Then, on the branch leading to $min(\mathcal{D})$ is the node corresponding to $\mathcal{D}'$: $min(\mathcal{D}')$. As $\mathcal{D}' = \mathcal{D}$, $min(\mathcal{D}') = min(\mathcal{D})$ and $min(\mathcal{D})$ is labelled by $\top$.

3. there is a *safe* valid set of actions $v \in valid(\mathbf{Enabled}^{symb}_{\mathcal{D}et}(\mathcal{D}))$.

   - If $min(\mathcal{D})$ (which cannot be bad) has no successor: then it is labelled by $\top$ in $T^=_{min}$ and we are done.
   - otherwise, we will use the fact that v is valid from min($\mathcal{D}$), i.e. $v \in valid(\mathbf{Enabled}^{symb}_{\mathcal{D}et}(min(\mathcal{D})))$ (see Lemma 7.61 and Remark 7.62). We will show that v is safe from $min(\mathcal{D})$ in $T^=_{min}$. Let $\mathcal{D}'$ be a successor of $\mathcal{D}$ by a symbolic action $(a, g, R) \in v$ in $T^=$: $\mathcal{D}'$ is labelled by $\top$ and has $min(\mathcal{D}')$ as corresponding node in $T^=_{min}$. We must show that $min(\mathcal{D}')$ is labelled by $\top$ in $T^=_{min}$. Once again, the same cases can be developed from corresponding nodes $\mathcal{D}'$ labelled by $\top$ in $T^=$ and

$min(\mathcal{D}')$ in $T^{=}_{min}$. As the current case can only be repeated a finite number of times, because $T^{=}$ and $T^{=}_{min}$ are finite, we are done.

($\Leftarrow$) Let us fix a sequence s of safe valid sets of actions, following a branch of $T^{=}_{min}$ from the root of $T^{=}_{min}$, witnessing that its root is labelled by $\top$. We will only consider the nodes of $T^{=}_{min}$ reached following s and call S_{min} the subtree of $T^{=}_{min}$ they form. All the nodes of S_{min} are of course labelled by $\top$ in $T^{=}_{min}$ and we will show their corresponding nodes in $T^{=}$ are also labelled by $\top$. This proves the root of $T^{=}$ will be labelled by $\top$, because, by Lemma 7.61, s is also a succession of valid sets of actions, from the root of $T^{=}$.
Let $min(\mathcal{D})$ be a node of S_{min}: $min(\mathcal{D})$ is labelled by $\top$. Let us note $\mathcal{D}$ its corresponding node in $T^{=}$, we will show that $\mathcal{D}$ is also labelled by $\top$. There are three possible reasons for which $min(\mathcal{D})$ has been labelled by $\top$:

1. $min(\mathcal{D})$ has no successor. Then, $\mathcal{D}$ has no successor (see proposition 7.55): it is labelled by $\top$.

2. there is $min(\mathcal{D}')$ on the branch leading to $min(\mathcal{D})$ such that $\min(\mathcal{D}') = \min(\mathcal{D})$. We first remark that $min(\mathcal{D}')$ is a node of S_{min} and is so labelled by $\top$. Then, $\min(\mathcal{D})$ $(= \min(\mathcal{D}'))$ is not bad and so $\mathcal{D}$ is not bad. We distinguish two cases. On the one hand, if $T^{=}$ is stopped in $\mathcal{D}$, this is because there is $\mathcal{D}'$ on the branch leading to $\mathcal{D}$ such that $\mathcal{D}' = \mathcal{D}$: $\mathcal{D}$ is so labelled by $\top$ and we are done. On the second hand, if $T^{=}$ is *not* stopped in $\mathcal{D}$, $\mathcal{D}$ has successors in $T^{=}$ and we must show there is a *safe* valid set of actions $v \in valid(\mathbf{Enabled}^{symb}_{\mathcal{D}et}(\mathcal{D}))$ (see Figure 7.12). Let $v \in valid(\mathbf{Enabled}^{symb}_{\mathcal{D}et}(min(\mathcal{D}')))$ be a *safe* valid set of actions from $min(\mathcal{D}')$ $(= \min(\mathcal{D}))$: such a v exists because $min(\mathcal{D}')$ is not a leaf of $T^{=}_{min}$ and is labelled by $\top$. v is also a valid set of actions from $\mathcal{D}$ (see Lemma 7.61). We first show that, in $T^{=}$, each successor of $\mathcal{D}$ by a symbolic action $(a, g, R) \in v$ has its minimal elements given by the successor of $min(\mathcal{D}')$, for the same action (a, g, R), in $T^{=}_{min}$. Let $\mathcal{D}$" be the successor of $\mathcal{D}$ reading $(a, g, R) \in v$ in $T^{=}$ and $m\mathcal{D}$" the successor of $min(\mathcal{D}')$ reading (a, g, R) in $T^{=}_{min}$. We

have:

$$\begin{aligned}\min(\mathcal{D}") &= \min(Post^{a,g,R}(\mathcal{D})) \\ &= \min(Post^{a,g,R}(\min(\mathcal{D}))) \quad \text{(Proposition 7.56)} \\ &= \min(Post^{a,g,R}(\min(\mathcal{D}'))) \\ &= m\mathcal{D}".\end{aligned}$$

Let us note S the subtree of $T^{=}$ whose root is $\mathcal{D}$ and leading only to valid sets of successors corresponding to safe valid sets of successors of the corresponding nodes in $T^{=}_{min}$. Then, following recursively the same reasoning, the sets of minimal elements of each node of S will be given by nodes of S_{min}. Remark that these nodes of S_{min} are not bad and labelled by $\top$. No node of S could so be labelled by $\bot$, because they could not be bad (as the sets of their minimal elements are not bad). As $T^{=}$ is finite, $\mathcal{D}$ will eventually be labelled by $\top$ thanks to this succession of safe valid sets of actions (leading only to nodes that could not be labelled by $\bot$ and will so eventually be labelled by $\top$).

3. there is a *safe* valid set of actions $v \in valid(\mathbf{Enabled}^{symb}_{\mathcal{D}et}(min(\mathcal{D})))$. By Lemma 7.61, $v \in valid(\mathbf{Enabled}^{symb}_{\mathcal{D}et}(\mathcal{D}))$. We will show that v is safe from $\mathcal{D}$ in $T^{=}$. Let $min(\mathcal{D}")$ be a successor of $min(\mathcal{D})$ by a symbolic action $(a, g, R) \in v$ in $T^{=}_{min}$ ($min(\mathcal{D}")$ is so labelled by $\top$) and has $\mathcal{D}"$ as corresponding node in $T^{=}$. We must show that $\mathcal{D}"$ is labelled by $\top$ in $T^{=}$. The three same cases can be developed from corresponding nodes $min(\mathcal{D}")$ labelled by $\top$ in $T^{=}_{min}$ and $\mathcal{D}"$ in $T^{=}$. As the current case can only be repeated a finite number of times as $T^{=}$ and $T^{=}_{min}$ are finite, we are done.

□

Combining both heuristics. Our following aim will be to use synchronously the order $\sqsubseteq_{p.s.}$ defined by Bouyer and al. in [17] and our own order (keeping $min(\mathcal{D})$ instead of each location $\mathcal{D}$ of $\mathcal{D}et$). We will prove that the algorithm of Definition 7.40 is still correct when using the order $\sqsubseteq_{p.s.}$ instead of $=$ to

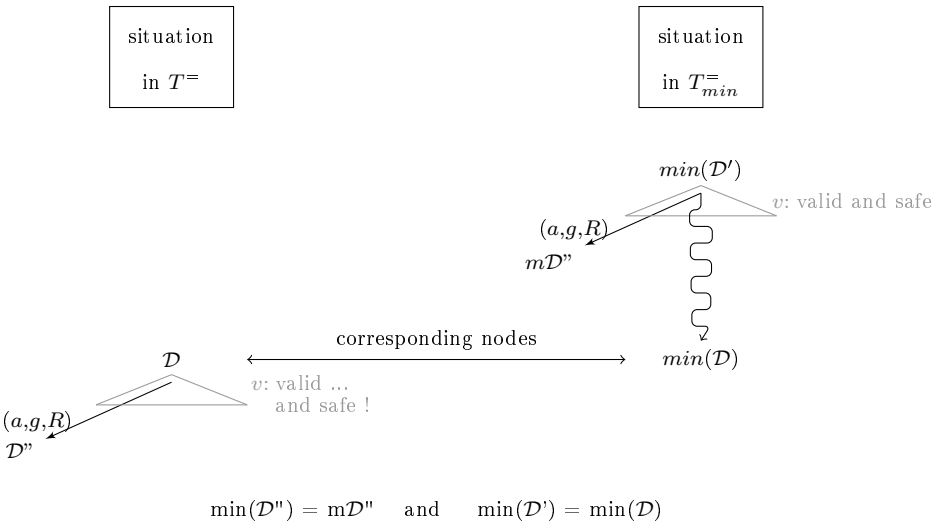


Figure 7.12: Representation of the situation of case 2., when $min(\mathcal{D})$ has no successor.

truncate the branches of the unfolding of $\mathcal{D}et$ and simultaneously conserving min($\mathcal{D}$) instead of each location $\mathcal{D}$ of $\mathcal{D}et$. Let us formally define the finite part of the unfolding T of $\mathcal{D}et$ constructed this way.

Definition 7.63. *We define $T^{\sqsubseteq}_{min}$ the finite part of the unfolding of $\mathcal{D}et = (SW, \{w_0\}, \Delta_{\mathcal{D}et})$ recursively obtained as follows:*

- *the root of $T^{\sqsubseteq}_{min}$ is the same as that of T.*
- *for each node $\mathcal{D} \in SW$ in $T^{\sqsubseteq}_{min}$, if the branch of $T^{\sqsubseteq}_{min}$ leading to $\mathcal{D}$ already contains another node $\mathcal{D}'$ such that $\mathcal{D}' \sqsubseteq \mathcal{D}$, then, $\mathcal{D}$ has no successor; otherwise, for each symbolic action $(a, g, R) \in \Sigma \times \mathcal{G}(X \cup X_{\mathcal{C}}) \times 2^{X \cup X_{\mathcal{C}}}$ such that $Post^{a,g,R}(\mathcal{D}) \neq \emptyset$, there is an arc labelled by (a, g, R) from $\mathcal{D}$ to $min(Post^{a,g,R}(\mathcal{D}))$.*

Our following aim will be to prove that labelling the nodes of $T^{\sqsubseteq}_{min}$ instead of those of $T^{\sqsubseteq}$ is correct in way to solve the MITL BRSPlant problem, as stated by the following theorem:

Theorem 7.64. *Using the procedure of the algorithm of Definition 7.40 to labelled the nodes of $T^{\sqsubseteq}$ and $T^{\sqsubseteq}_{min}$, we have that:*

$$\textit{the root of } T^{\sqsubseteq} \textit{ is labelled by } \top \quad \textit{iff} \quad \textit{the root of } T^{\sqsubseteq}_{min} \textit{ is labelled by } \top.$$

In way to prove this theorem, we start by expounding several kind properties of $T^{\sqsubseteq}_{min}$ and $T^{\sqsubseteq}$.

First of all, as in the case of $T^{=}$ and $T^{=}_{min}$, each branch of $T^{\sqsubseteq}_{min}$ is necessarily shorter than the corresponding branch in $T^{\sqsubseteq}$ but the converse is not true.

Proposition 7.65. Each branch of $T^{\sqsubseteq}_{min}$ is necessarily shorter than the corresponding branch in $T^{\sqsubseteq}$.

Proof. If a branch of $T^{\sqsubseteq}$ is stopped in a node $\mathcal{D}$, that has $min(\mathcal{D})$ as corresponding node in $T^{\sqsubseteq}_{min}$, there are three possible reasons for this:

1. $\mathcal{D}$ has no successor. Then, $min(\mathcal{D})$ and has no successor, by Proposition 7.55.

2. there is $\mathcal{D}'$ on the branch leading to $\mathcal{D}$ such that $\mathcal{D} \sqsubseteq_{p.s.} \mathcal{D}'$. Then, on the branch leading to $min(\mathcal{D})$ is the node corresponding to $\mathcal{D}'$ $\min(\mathcal{D}')$. We will prove that, as $\mathcal{D}' \sqsubseteq_{p.s.} \mathcal{D}$, we have that $min(\mathcal{D}') \sqsubseteq_{p.s.} min(\mathcal{D})$, what proves that $T^{\sqsubseteq}_{min}$ is stopped in $min(\mathcal{D})$. Let $w \in min(\mathcal{D})$, we must find $w' \in min(\mathcal{D}')$ such that $w' \sqsubseteq w$. As $\mathcal{D}' \sqsubseteq_{p.s.} \mathcal{D}$, we know there is $w^\star \in \mathcal{D}'$ such that $w^\star \sqsubseteq w$. If $w^\star \in \min(\mathcal{D}')$, we are done. Otherwise, there is $mw^\star \in \min(\mathcal{D}')$ such that $mw^\star \sqsubseteq w^\star$. So, $mw^\star \sqsubseteq w^\star \sqsubseteq w$ and $mw^\star$ is the w' we were looking for.

3. $\mathcal{D}$ is bad. Then, there is $w \in \mathcal{D}$ such that w is bad. As $min(\mathcal{D}) \sqsubseteq_{p.s.} \mathcal{D}$ and $w \in \mathcal{D}$, there is $mw \in min(\mathcal{D})$ such that $mw \sqsubseteq w$. As w is bad (all the locations of $\mathcal{B}$ and $\mathcal{A}_{\neg\Phi}$ present in w are accepting ones), mw is bad and so is $min(\mathcal{D})$. $T^{\sqsubseteq}_{min}$ is so stopped in $min(\mathcal{D})$.

□

Remark 7.66. $T^{\sqsubseteq}$ and $T^{\sqsubseteq}_{min}$ are *not* isomorphic because the reverses of cases 2. and 3. are not true in general.

Remark 7.67. By definition of $T^{\sqsubseteq}$ and $T^{\sqsubseteq}_{min}$ and Proposition 7.55, each node of $T^{\sqsubseteq}_{min}$ has: either no successor, either as many successors as its corresponding node in $T^{\sqsubseteq}$.

Another interesting property of $T^{\sqsubseteq}_{min}$ is that it is a subtree of $T^{=}_{min}$.

Proposition 7.68. $T^{\sqsubseteq}_{min}$ is a subtree of $T^{=}_{min}$.

Proof. Suppose a common branch of $T^{=}_{min}$ and $T^{\sqsubseteq}_{min}$ is stopped in a node $min\mathcal{D}$, in $T^{=}_{min}$: we will show it is also stopped in $T^{\sqsubseteq}_{min}$. There are three possible reasons for why $\mathcal{D}$ has no son in $T^{=}_{min}$:

1. $min\mathcal{D}$ has no successor: then, $T^{\sqsubseteq}_{min}$ is also stopped in $\mathcal{D}$;
2. there is $\min \mathcal{D}'$, on the branch leading to $min(\mathcal{D})$ such that $\min \mathcal{D} = \min \mathcal{D}'$. In particular, $\min \mathcal{D} \sqsubseteq \min \mathcal{D}'$ and $T^{\sqsubseteq}_{min}$ is also stopped in $min(\mathcal{D})$.
3. $\min \mathcal{D}$ is bad: the $T^{\sqsubseteq}_{min}$ is also stopped in $\min \mathcal{D}$.

□

Figure 7.13 summarizes the relative inclusions of T, $T^{=}$, $T^{\sqsubseteq}$ and the relative inclusions of $T^{=}_{min}$ and $T^{\sqsubseteq}_{min}$ in those trees, according to isomorphism. This figure shows the more efficient tree we want to apply the algorithm of Definition 7.40 to is $T^{\sqsubseteq}_{min}$. The following theorem shows the algorithm of Definition 7.40 (proved to be correct on $T^{\sqsubseteq}$ in [17]) is still correct when applied on $T^{\sqsubseteq}_{min}$ instead of $T^{\sqsubseteq}$.

Proof of Theorem 7.64. ($\Rightarrow$) The proof is similar to that of ($\Rightarrow$) in the proof of Theorem 7.50 using the fact that $\mathcal{D}' \sqsubseteq_{p.s.} \mathcal{D}$ implies $\min(\mathcal{D}') \sqsubseteq_{p.s.} \min(\mathcal{D})$

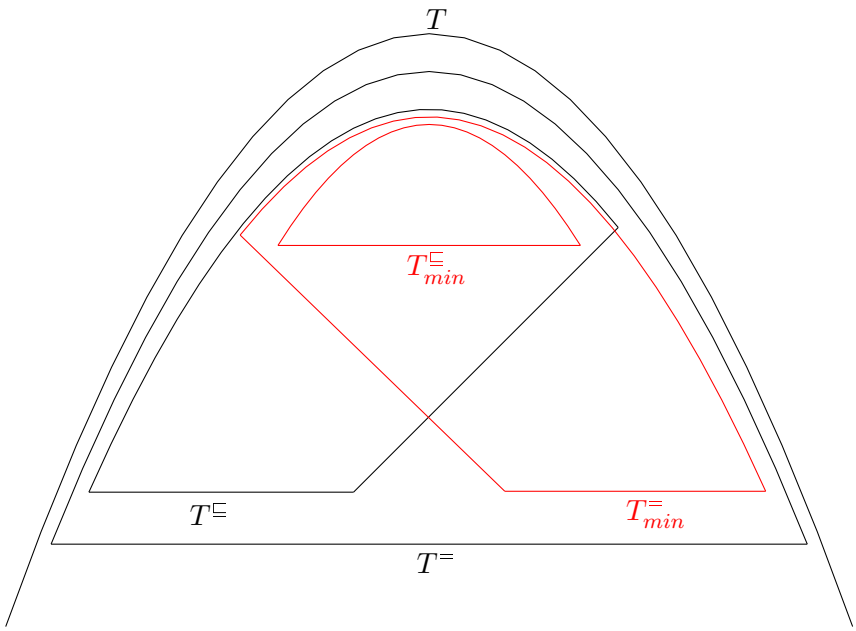


Figure 7.13: Relative inclusions of T, $T^{=}$, $T^{\sqsubseteq}$ and, according to isomorphism, of $T^{=}_{min}$ and $T^{\sqsubseteq}_{min}$.

(proven above).

($\Leftarrow$) The proof is similar to that of ($\Leftarrow$) in Theorem 7.50. Only the case 2. needs a new argument. Let $\min(\mathcal{D})$, a node of $T^{\sqsubseteq}_{min}$ labelled by $\top$, and $\mathcal{D}$, its corresponding node in $T^{\sqsubseteq}$. We will show that $\mathcal{D}$ is also labelled by $\top$ when (case 2.) there is $\min(\mathcal{D}')$, on the branch leading to $min(\mathcal{D})$ such that $\min(\mathcal{D}) \sqsubseteq_{p.s.} \min(\mathcal{D}')$. To prove this, we show that $\min(\mathcal{D}) \sqsubseteq_{p.s.} \min(\mathcal{D}')$ implies $\mathcal{D} \sqsubseteq_{p.s.} \mathcal{D}'$, so that the branch of $T^{\sqsubseteq}$ is stopped in $\mathcal{D}$ and $\mathcal{D}$ is labelled by $\top$. We suppose $\min(\mathcal{D}) \sqsubseteq_{p.s.} \min(\mathcal{D}')$. Let $w' \in \mathcal{D}'$, we are looking for $w \in \mathcal{D}$ such that $w \sqsubseteq w'$. As $\min(\mathcal{D}') \sqsubseteq_{p.s.} \mathcal{D}'$, there is $\tilde{w}' \in \min(\mathcal{D}')$ such that $\tilde{w}' \sqsubseteq w'$. As $\min(\mathcal{D}) \sqsubseteq_{p.s.} \min(\mathcal{D}')$, there is $\tilde{w} \in \min(\mathcal{D})$ such that $\tilde{w} \sqsubseteq \tilde{w}' \sqsubseteq w'$. So, $\tilde{w} \sqsubseteq w'$ and as $\min(\mathcal{D}) \subseteq \mathcal{D}$, $\tilde{w}$ is the w we were looking for. $\square$

CHAPTER 8

Experimental results

We implemented the algorithm composed of the different steps presented in the previous chapter to solve the MITL BRSPlant problem. This chapter is dedicated to the results obtained when testing this program. As for the MITL satisfiability and model-checking algorithms, we performed our tests on a Mac Pro (mid 2010) with OS X Yosemite, processor 3.33 GHz, 6 core intel xeon, with a memory of 32 GB 1333 MHz DDR3 ECC. We used Java SE Runtime Environment (build 1.6.0_65-b14-466.1-11M4716).

We notice that the algorithm presented in the previous chapter was cleverly implemented. Indeed, the reachable locations of $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$ (see Definition 7.27) and $\mathcal{D}et$ (see Definition 7.33) are computed *on the fly*, as well as the labellings by $\top$ or $\bot$ of the finite constructed portion of the tree given by the unfolding of $\mathcal{D}et$. We tested our program on two benchmarks.

The first benchmark consists in a scheduling problem adapted from [23]. This problem considers n computational units $u_1, \ldots, u_n$. Some *job*'s must be realized: each time a *job* arrives, a computational unit must be assigned to conduct it. A

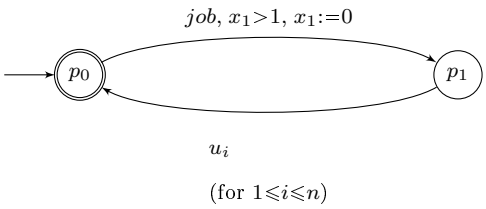


Figure 8.1: The plant $\mathcal{P}$ used to represent the scheduling problem.

job takes T time units to be realized. Our first hypothesis, ensured by the plant, is that the minimal time between the arrivals of two *job*'s (an uncontrollable action) is 1 time unit. The plant we use is presented in Figure 8.1. The specifications we consider always have the same form. First, a job arrival (*job*, an action of the environment) must be followed, in less than 1 time unit, by the assignment of a computational unit (u_1 or ... or u_n, actions of the controller) to realize it, and second, when a computational unit u_i has just been assigned a job, it cannot be assigned a new job for T time units. Precisely, we considered the cases $T = n$, for $1 \leqslant T \leqslant 5$, and $T = n + 1$, for $2 \leqslant T \leqslant 5$. When $T = n$, the controller has a winning strategy to ensure the satisfaction of the specification, not when $T = n + 1$. For the granularity of the controller, we always keep the maximal constant allowed to be that of the considered formula. However, for each formula, we consider granularities giving 0, 1 and finally 2 clocks to the controller.

Tables 8.1 and 8.2 report on results over four versions of the MITL BRSPlant algorithm. Table 8.1 concerns controllable formulas while Table 8.2 is dedicated to uncontrollable formulas. These tables contains the following columns:

- column 'T' gives the number T such that a *job* takes T time units to be realized,
- column 'n' gives the number of compositional units,
- column 'Clocks?' giving the number of clocks the controller is allowed to use,

- column $T^{=}$ (respectively $T^{\sqsubseteq}$, $T^{=}_{min}$ and $T^{\sqsubseteq}_{min}$) is dedicated to the execution of our MITL BRSPlant algorithm executed on the finite portion of the tree given by the unfolding of $\mathcal{D}et$ in which branches are stopped using $=$ (respectively, in which branches are stopped using the order $\sqsubseteq$, in which only minimal elements are kept and branches are stopped using $=$, and in which only minimal elements are kept and branches are stopped using $\sqsubseteq$).

Columns $T^{=}$, $T^{\sqsubseteq}$, $T^{=}_{min}$ and $T^{\sqsubseteq}_{min}$ contain the execution time, followed by the number of constructed locations of $\mathcal{D}et$ / of $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$. A time out was set after 5 minutes.

Table 8.1: Scheduling problem - Controllable formulas. Reported values are execution time in ms, followed by the number of constructed locations of $\mathcal{D}et$ / of $\mathcal{ST}^{\equiv}_{\overline{\mathcal{P}},\neg\Phi}$.

T	n	Formula	Clocks?	$T^{=}$	$T^{=}_{min}$	$T^{\sqsubseteq}$	$T^{\sqsubseteq}_{min}$
1	1	$\Box(\,(job \Rightarrow \Diamond_{[0,1[}u_1)$	0	46	55	48	57
		$\wedge\,(u_1 \Rightarrow \Box_{]0,1]}\neg u_1)\,)$		16 / 52	16 / 52	16 / 52	16 / 52
			1	199	405	193	268
				47 / 147	47 / 147	47 / 147	47 / 147
			2	4,599	4,648	4,688	4,584
				554 / 1,343	554 / 1,343	544 / 1,260	544 / 1,260
2	2	$\Box(\,\left(job \Rightarrow \bigvee_{i=1}^{2} \Diamond_{[0,1[}u_i\right)$	0	503	576	535	554
		$\wedge \bigwedge_{i=1}^{2} (u_i \Rightarrow \Box_{]0,2]}\neg u_i)\,)$		138 / 149	138 / 149	128 / 149	128 / 149
			1	2,632	4,386	2,290	2,584
				719 / 645	719 / 645	580 / 620	642 / 623
			2	18,453	18,075	15,910	16,276
				1,957 / 2,358	1,896 / 2,310	1,683 / 2,197	1,683 / 2,197
3	3	$\Box(\,\left(job \Rightarrow \bigvee_{i=1}^{3} \Diamond_{[0,1[}u_i\right)$	0	4,210	4,706	4,318	4,477
		$\wedge \bigwedge_{i=1}^{3} (u_i \Rightarrow \Box_{]0,3]}\neg u_i)\,)$		983 / 342	1,018 / 342	988 / 342	988 / 342
			1	182,524	189,177	178,256	184,627
				19,871 / 2,297	19,250 / 2,385	19,806 / 2,297	18,805 / 2,385
			2	>5min	>5min	>5min	>5min
4	4	$\Box(\,\left(job \Rightarrow \bigvee_{i=1}^{4} \Diamond_{[0,1[}u_i\right)$	0	54,893	72,625	64,606	76,250
		$\wedge \bigwedge_{i=1}^{4} (u_i \Rightarrow \Box_{]0,4]}\neg u_i)\,)$		11,000 / 667	11,282 / 667	11,131 / 667	11,020 / 667
			1	>5min	>5min	>5min	>5min
			2	>5min	>5min	>5min	>5min
5	5	$\Box(\,\left(job \Rightarrow \bigvee_{i=1}^{5} \Diamond_{[0,1[}u_i\right)$	0	>5min	>5min	>5min	>5min
		$\wedge \bigwedge_{i=1}^{5} (u_i \Rightarrow \Box_{]0,5]}\neg u_i)\,)$					
			1	>5min	>5min	>5min	>5min
			2	>5min	>5min	>5min	>5min

Table 8.2: Scheduling problem - Uncontrollable formulas. Reported values are execution time in ms, followed by the number of constructed locations of $\mathcal{D}et$ / of $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$.

T	n	Formula	Clocks?	$T^{=}$	$T^{=}_{min}$	$T^{\sqsubseteq}$	$T^{\sqsubseteq}_{min}$
2	1	$\Box(\,(job \Rightarrow \Diamond_{[0,1[}u_1)$	0	77	99	77	101
		$\wedge\,(u_1 \Rightarrow \Box_{]0,2]}\neg u_1)\,)$		25 / 84	25 / 84	25 / 84	25 / 84
			1	824	895	827	889
				127 / 311	127 / 311	127 / 311	127 / 311
			2	3,079	3,263	3,118	3,108
				408 / 1,116	421 / 1,072	408 / 1,116	408 / 1,116
3	2	$\Box(\,\left(job \Rightarrow \bigvee_{i=1}^{2} \Diamond_{[0,1[}u_1\right)$	0	659	913	667	926
		$\wedge \bigwedge_{i=1}^{2} (u_i \Rightarrow \Box_{]0,3]}\neg u_i)\,)$		268 / 254	288 / 254	268 / 254	288 / 254
			1	17134	21018	19151	22041
				3380 / 1698	3734 / 1777	3789 / 1741	4271 / 1806
			2	>5min	>5min	>5min	>5min
4	3	$\Box(\,\left(job \Rightarrow \bigvee_{i=1}^{3} \Diamond_{[0,1[}u_1\right)$	0	10,621	12,478	7,657	8,298
		$\wedge \bigwedge_{i=1}^{3} (u_i \Rightarrow \Box_{]0,4]}\neg u_i)\,)$		3018 / 540	3,018 / 540	2,094 / 492	2,009 / 436
			1	>5min	>5min	>5min	>5min
			2	>5min	>5min	>5min	>5min
5	4	$\Box(\,\left(job \Rightarrow \bigvee_{i=1}^{4} \Diamond_{[0,1[}u_1\right)$	0	>5min	>5min	>5min	>5min
		$\wedge \bigwedge_{i=1}^{4} (u_i \Rightarrow \Box_{]0,5]}\neg u_i)\,)$					
			1	>5min	>5min	>5min	>5min
			2	>5min	>5min	>5min	>5min

Observing results of Tables 8.1 and 8.2, we first remark that the use of the order $\sqsubseteq$, as well as the idea to only keep the *minimal* elements of locations of $\mathcal{D}et$, were not fruitful. Indeed, the number of constructed locations of $\mathcal{D}et$ / $\mathcal{ST}^{=}_{\mathcal{P},\neg\Phi}$ are very similar (and most of the time equal) when using $T^{=}$, $T^{=}_{min}$, $T^{\sqsubseteq}$ or $T^{\sqsubseteq}_{min}$. However, the execution time on $T^{=}_{min}$, $T^{\sqsubseteq}$ and $T^{\sqsubseteq}_{min}$ are in general higher than that on $T^{=}$. This is probably due to the supplementary computation performed by the program to keep the antichain of minimal elements or / and to compare the constructed locations using $\sqsubseteq$. Precisely, the execution time is increased, in average, by 15% from $T^{=}$ to $T^{=}_{min}$, by 5% from $T^{=}$ to $T^{\sqsubseteq}$ and by 20% from $T^{=}$ to $T^{\sqsubseteq}_{min}$. Nevertheless, on some examples, using $T^{\sqsubseteq}$ instead of $T^{=}$ reduces the execution time by 2%, 20% and even 43%.
We also notice that the program quickly struggles, for little values of the parameters T and n.

As a second benchmark, we once again consider the family of timed automata $\mathcal{B}_k^{lift}$, modelling a *lift*, parametrised by its number k of floors (see Chapter 6 for details). This time, each $\mathcal{B}_k^{lift}$ is considered to be a plant. Two letters are now consider as uncontrollable actions and form the alphabet of the environment:

- letter l_i, meaning that the button of floor i has just been pushed to make the lift come at this floor, and
- letter b_i, meaning that the button present in the lift cabin to send it at floor i has just been pushed.

The remaining letters are considered to be controllable actions and form the alphabet of the controller:

- letter o_i and, respectively, letter c_i, signifying the lift opens and, respectively, closes its doors at floor i;
- letter p_i, meaning the lift passes floor i without stopping.

The aim of the controller is to ensure the lift eventually reaches a floor i by a 'brief' delay, each time l_i or b_i has been pushed.

The obtained results were reported in Table 8.3, which contains the same kind of columns as Tables 8.1 and 8.2. It additionnaly contains a column 'Floors' giving the number of floors of the considered lift and a column 'Contr?' giving the expected[1] answer.

We got very few results on this example (even giving no clock to the controller), as witnesses Table 8.3. We notice that our program has facilities to prove that a formula in uncontrollable. However, it only succeeded to prove (by 5 minutes) that one formula is controllable among all those proposed.

[1] and obtained, when the program terminates on this test !

Table 8.3: Lift problem. Reported values are execution time in ms, followed by the number of constructed locations of $\mathcal{D}et$ / of $\mathcal{ST}^{\equiv}_{\mathcal{P},\neg\Phi}$.

Floors	Formula	Contr?	Clocks?	$T^{=}$	$T^{=}_{min}$	$T^{\sqsubseteq}$	$T^{\sqsubseteq}_{min}$
2	$\Box\left(\bigwedge_{i=0}^{1}(l_i \Rightarrow \Diamond_{[0,6]}o_i)\right)$	×	0	23	18	16	14
				3 / 3	3 / 3	3 / 3	3 / 3
2	$\Box\left(\bigwedge_{i=0}^{1}((l_i \wedge \Diamond_{[0,+\infty[}o_i) \Rightarrow \Diamond_{[0,6]}o_i)\right)$	✓	0	>5min	>5min	>5min	>5min
2	$\Box((l_0 \wedge \Diamond_{[0,+\infty[}o_0) \Rightarrow \Diamond_{[0,6]}o_0)$	✓	0	118,795	117,415	139,130	100,534
				26,837 / 1,667	24,596 / 1,633	29,583 / 1,744	21,742 / 1,595
3	$\Box\left(\bigwedge_{i=0}^{2}(l_i \Rightarrow \Diamond_{[0,14]}o_i)\right)$	×	0	125	99	90	100
				5 / 5	5 / 5	5 / 5	5 / 5
3	$\Box\left(\bigwedge_{i=0}^{2}((l_i \wedge \Diamond_{[0,+\infty[}o_i) \Rightarrow \Diamond_{[0,14]}o_i)\right)$	✓	0	>5min	>5min	>5min	>5min
3	$\Box((l_0 \wedge \Diamond_{[0,+\infty[}o_0) \Rightarrow \Diamond_{[0,14]}o_0)$	✓	0	>5min	>5min	>5min	>5min
4	$\Box\left(\bigwedge_{i=0}^{3}(l_i \Rightarrow \Diamond_{[0,22]}o_i)\right)$	×	0	473	212	86	88
				7 / 7	7 / 7	7 / 7	7 / 7
4	$\Box\left(\bigwedge_{i=0}^{3}((l_i \wedge \Diamond_{[0,+\infty[}o_i) \Rightarrow \Diamond_{[0,22]}o_i)\right)$	✓	0	>5min	>5min	>5min	>5min
4	$\Box((l_0 \wedge \Diamond_{[0,+\infty[}o_0) \Rightarrow \Diamond_{[0,22]}o_0)$	✓	0	>5min	>5min	>5min	>5min
5	$\Box\left(\bigwedge_{i=0}^{3}(l_i \Rightarrow \Diamond_{[0,30]}o_i)\right)$	×	0	840	238	159	1186
				9 / 9	9 / 9	9 / 9	9 / 9
5	$\Box\left(\bigwedge_{i=0}^{3}((l_i \wedge \Diamond_{[0,+\infty[}o_i) \Rightarrow \Diamond_{[0,30]}o_i)\right)$	✓	0	>5min	>5min	>5min	>5min
5	$\Box((l_0 \wedge \Diamond_{[0,+\infty[}o_0) \Rightarrow \Diamond_{[0,30]}o_0)$	✓	0	>5min	>5min	>5min	>5min5

CHAPTER 9

Conclusion and future work

All along this thesis, we investigated decidable verification problems about MITL, aiming to obtain algorithms which are efficient in practice.

The first part of this thesis was dedicated to the MITL satisfiability and model-checking by means of OCATA. Thanks to a new semantics for OCATA, we presented a new algorithm to construct from any MITL formula Φ a timed automaton $\mathcal{B}_\Phi$ recognizing $[\![\Phi]\!]$, and, over infinite words, a Büchi timed automaton $\mathcal{B}_\Phi^\omega$ recognizing $[\![\Phi]\!]^\omega$. We also presented algorithms working *on the fly* to solve the MITL model-checking problem. Our first algorithms were based on a region-based abstraction, and the second ones on a zone-based abstraction. Over finite words, we also give heuristics of these region-based and zone-based algorithms using antichains. All these algorithms were implemented in a Java prototype. The tests we performed on our programs show that the use of zones is more efficient than that of regions: it enables to obtain the answer more quickly and computing less zones than the number of regions calculated with the other algorithm. The results of tests are less marked on the efficiency of the use of antichains. The number of explored regions / zones are lower when using them,

but the execution time stay unchanged, probably because of the supplementary computation needed in the program to maintain those antichains. Nevertheless, our program enabled to answer various satisfiability and model-checking problems of reasonable size. As far as we know, our prototype of tools is the first implementation of an MITL model-checker.

In the second part of this thesis, we investigated the MITL BRSPlant problem. We based our work on the existing results over the MTL BRSPlant problem. Thanks to the use of our new semantics for OCATA, we showed that the existing non-primitive recursive algorithm for the MTL BRSPlant problem could be improved for the particular case of MITL. Indeed, we obtained an MITL BRSPlant algorithm executing in time triply exponential in the sizes of the considered MITL formula and plant. Then, we presented two orders leading to three possible heuristics for our basis algorithm. We also implemented these algorithms in a Java prototype. Unfortunately, this program quickly encounters difficulties solve problems, even on small instances. The heuristics based on antichains do not seem to help it to be more efficient.

Let us now present some future investigation directions. From a practical point of view, it would be interesting to test our prototype of tool for the MITL satisfiability, model-checking and BRSPlant problems on supplementary benchmarks. In particular, we could try to develop benchmarks on which our MITL BRSPlant algorithm is more efficient. Another possible research direction could consist in the improvement of all the implemented algorithms (with a particular attention to the MITL BRSPlant algorithm). The algorithms implemented to solve the LTL reactive synthesis were recently improved thanks to the use of adapted data structures [14]: such a trail should be explored. From a theoretical point of view, we are particularly insterested in the MITL BRSPlant problem. We recall this problem consists in fixing *a priori* the number of clocks the controller is allowed to use to construct a winning strategy against the environment. However, when considering an MITL formula Φ, we do not see how the controller could need more clocks than the number of clocks necessary to verify that Φ is

satisfied (i.e. $M(\Phi)$ clocks). We may think that it is not necessary to bound *a priori* the number of clocks the controller is allowed to use in way to keep the decidability of the MITL BRSPlant problem: this bound should exist anyway and be $M(\Phi)$.

APPENDIX A

Proofs of Propositions 4.8 and 4.9

Let us recall useful results from [51] that enable to prove Proposition 4.8 and Proposition 4.9.

Proposition A.1 ([51])**.** Let $\Phi \equiv \varphi_1 U_I \varphi_2$ or $\Phi \equiv \varphi_1 \tilde{U}_I \varphi_2$ be an MITL formula and ℓ_Φ the associated location in $\mathcal{A}_\Phi$. Let θ be an infinite timed word.

$$\text{The automaton } \mathcal{A}_\Phi \text{ accepts } \theta \text{ from configuration } \{(\ell_\Phi, 0)\} \text{ iff } \theta \models \Phi.$$

The following corollary directly follows from Proposition A.1.

Corollary A.2. *Let Φ be an MITL formula, let K be a set of indices and, $\forall k \in K$ let $\Phi_k = \varphi_{1,k} U_{I_k} \varphi_{2,k}$ or $\Phi_k \varphi_{1,k} \tilde{U}_{I_k} \varphi_{2,k}$ be subformulas of Φ. For all $k \in K$, let ℓ_{Φ_k} the associated locations in $\mathcal{A}_\Phi$. Let θ be an infinite timed word and, for all $k \in K$. Then:*

$$\mathcal{A}_\Phi \textit{ accepts } \theta \textit{ from configuration } \{(\ell_{\Phi_k}, 0)_{k \in K}\} \textit{ iff } \theta \models \bigwedge_{k \in K} \varphi_{1,k} U_{I_k} \varphi_{2,k}.$$

Let us now adapt this result to the cases where the automaton reads the word from states of the form (ℓ, v), with v potentially $\neq 0$:

Lemma A.3. *Let Φ be an MITL formula, let K be a set of indices and, $\forall k \in K$ let $\Phi_k = \varphi_{1,k} U_{I_k} \varphi_{2,k}$ or $\Phi_k = \varphi_{1,k} \tilde{U}_{I_k} \varphi_{2,k}$ be subformulas of Φ. For all $k \in K$, let ℓ_{Φ_k} be their associated locations in $\mathcal{A}_\Phi$. Let θ be an infinite timed word and $v_k \in \mathbb{R}^+$ $(\forall k \in K)$.*

$$\textit{The automaton } \mathcal{A}_\Phi \textit{ accepts } \theta \textit{ from configuration } \{(\ell_{\Phi_k}, v_k)_{k \in K}\} \textit{ iff } \theta \models \bigwedge_{k \in K} (\Phi_k - v_k),$$

where, for all $k \in K$, $\Phi_k - v_k$ denotes the formula obtained from Φ_k by replacing the I_k interval on the modality by $I_k - v_k$.

Proof. We prove that, for all $k \in K$ such that the outer modality of Φ_k is U: $\mathcal{A}_\Phi$ accept θ from (ℓ_{Φ_k}, v_k) iff $\theta \models \varphi_{1,k} U_{I_k - v_k} \varphi_{2,k}$. The same arguments adapt to the $\tilde{U}$ case, and the lemma follows.

Assume $\theta = (\overline{\sigma}, \overline{\tau})$ with $\overline{\sigma} = \sigma_1 \sigma_2 \ldots \sigma_i \ldots$ and $\overline{\tau} = \tau_1 \tau_2 \ldots \tau_i \ldots$. For all $v \in \mathbb{R}^+$, let $\theta + v$ denote the infinite timed word $(\overline{\sigma}, \tau')$, where $\tau' = (\tau_1 + v)(\tau_2 + v) \ldots (\tau_i + v) \ldots$. Observe that, by definition of the semantics of MITL,

$$\theta \models \varphi_1 U_I \varphi_2 \text{ iff } \theta + v \models \varphi_1 U_{I+v} \varphi_2$$

(remark that the φ_1 and φ_2 formulas are preserved).

First, assume that $\theta \models \varphi_{1,k} U_{I_k - v_k} \varphi_{2,k}$ and let us show that $\mathcal{A}_\Phi$ accepts θ from (ℓ_{Φ_k}, v_k). Since $\theta \models \varphi_{1,k} U_{I_k - v_k} \varphi_{2,k}$, $\theta + v_k \models \varphi_{1,k} U_{I_k} \varphi_{2,k} \equiv \Phi_k$. Then, by Proposition A.1, $\mathcal{A}_\Phi$ accepts $\theta + v_k$ from $(\ell_{\Phi_k}, 0)$. Let $(\ell_{\Phi_k}, 0) \xrightarrow{\tau_1 + v_k, \sigma_1} C_1 \xrightarrow{\tau_2 - \tau_1, \sigma_2} \cdots \xrightarrow{\tau_i - \tau_{i-1}, \sigma_i} C_i \ldots$ be an accepting run of $\mathcal{A}_\Phi$ from $(\ell_{\Phi_k}, 0)$ on $\theta + v_k$. Obviously, the first time step can be decomposed as follows:

$$(\ell_{\Phi_k}, 0) \overset{v_k}{\rightsquigarrow} (\ell_{\Phi_k}, v_k) \xrightarrow{\tau_1, \sigma_1} C_1 \xrightarrow{\tau_2 - \tau_1, \sigma_2} \cdots \xrightarrow{\tau_i - \tau_{i-1}, \sigma_i} C_i \ldots,$$

where the suffix starting in (ℓ_{Φ_k}, v_k) is an accepting run on θ. We conclude that $\mathcal{A}_\Phi$ accepts θ from (ℓ_{Φ_k}, v_k).

By using the same arguments, we can prove that $\mathcal{A}_\Phi$ accepts θ from (ℓ_{Φ_k}, v_k) *implies* that $\theta \models \varphi_{1,k} U_{I_k - v_k} \varphi_{2,k}$. □

We will also need this simple lemma:

Lemma A.4. *Let Φ be an MITL formula, let K a set of indices and, $\forall k \in K$, let Φ_k be subformulas of Φ of the form either $\varphi_{1,k} U_{I_k} \varphi_{2,k}$ or $\varphi_{1,k} \tilde{U}_{I_k} \varphi_{2,k}$. For all $k \in K$, let ℓ_{Φ_k} be their associated locations in $\mathcal{A}_\Phi$. Let θ be an infinite timed word and $J_k \in \mathcal{I}(\mathbb{R}^+)$ $(\forall k \in K)$.*

The automaton $\mathcal{A}_\Phi$ accepts θ from configuration $\{(\ell_{\Phi_k}, J_k)_{k \in K}\}$
iff
$\forall k \in K, \mathcal{A}_\Phi$ accepts θ from configuration $\{(\ell_{\Phi_k}, J_k)\}$.

Proof. It is straightforward by definition of runs on $\mathcal{A}_\Phi$: the time elapsed is reported on each state (ℓ_{Φ_k}, J_k) and the reading of a letter gives a minimal model for each state (ℓ_{Φ_k}, J_k) before to merge them into a unique new configuration. □

Now, we recall a second result from [51]:

Lemma A.5 ([51])**.** *Let Φ be an MITL formula and φ a subformula of Φ. Let $\theta = (\overline{\sigma}, \overline{\tau})$ be a finite timed word and π: $C_0 = \{(\ell_\Phi, 0)\} \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \ldots \overset{t_n}{\rightsquigarrow} C_{2n-1} \xrightarrow{\sigma_n} C_{2n}$ an accepting run of $\mathcal{A}_\Phi$ on θ. For all $1 \leqslant i \leqslant n$:*

$$C_{2i} \models_{[0,0]} \delta(\varphi, \sigma_i) \ \ \textit{implies} \ \ (\theta, i) \models \varphi.$$

Thanks to a proof very similar to that of the previous Lemma (as presented in Proposition 6.4 in [51]), we obtain the similar following result over *infinite words*:

Lemma A.6. *Let Φ be an MITL formula and φ a subformula of Φ. Let $\theta = (\overline{\sigma}, \overline{\tau})$ be an infinite timed word and π: $C_0 = \{(\ell, J)\} \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \dots \overset{t_i}{\rightsquigarrow} C_{2i-1} \xrightarrow{\sigma_i} C_{2i} \dots$ an accepting run of $\mathcal{A}_\Phi$ on θ from $\{(\ell, J)\}$. For all $i \geqslant 1$:*

$$C_{2i} \models_{[0,0]} \delta(\varphi, \sigma_i) \ \ \textit{implies} \ \ (\theta, i) \models \varphi.$$

We can now prove Propositions 4.8 and 4.9. We here present the proofs over *infinite timed words*, but they are identical for the setting of finite timed words. In these proofs, we will use the following notations. $\theta = (\overline{\sigma}, \overline{\tau})$ is a timed word with $\overline{\sigma} = \sigma_1\sigma_2 \dots \sigma_i \dots$ and $\overline{\tau} = \tau_1\tau_2 \dots \tau_i \dots$. For all $k \geqslant 1$, we denote by $\theta^k = (\overline{\sigma}_k, \overline{\tau}_k)$, where $\overline{\sigma}_k = \sigma_k\sigma_{k+1} \dots \sigma_i \dots$ and $\overline{\tau}_k = \tau'_1\tau'_2 \dots \tau'_{i-k} \dots$, the infinite timed word such that $\forall i \geqslant 1, \tau'_i = \tau_{i+k} - \tau_k$.
We moreover recall the following notation:

$$\mho(k, i, I, J) :=$$
$$(\theta^k, i) \models \Phi_2 \wedge \tau_i^k \in I - \inf(J) \wedge \tau_i^k \in I - \sup(J) \wedge \forall 1 \leqslant m' < i : (\theta^k, m') \models \Phi_1.$$

Proposition 4.8: *Let Φ be an MITL formula, let K be a set of indices and, $\forall k \in K$, let $\Phi_k = \Phi_{1,k} U_{I_k} \Phi_{2,k}$ be subformulas of Φ. For all $k \in K$, let ℓ_{Φ_k} be their associated locations in $\mathcal{A}_\Phi$. Let $\theta = (\overline{\sigma}, \overline{\tau})$ be an infinite timed word and let $J_k \in \mathcal{I}(\mathbb{R}^+)$ be closed intervals.*

The automaton $\mathcal{A}_\Phi$ accepts θ from configuration $\{(\ell_{\Phi_k}, J_k)_{k \in K}\}$
iff
$$\forall k \in K, \exists m_k \geqslant 1 : \mho(0, m_k, I, J).$$

Proof. Thanks to Lemma A.4, we only need to prove the following. Let Ψ be an MITL formula, $\Phi := \varphi_1 U_I \varphi_2$ a subformula of Ψ and ℓ_Φ its associated location in $\mathcal{A}_\Psi$. Let $\theta = (\overline{\sigma}, \overline{\tau})$ be an infinite timed word and $J \in \mathcal{I}(\mathbb{R}^+)$ closed.

The automaton $\mathcal{A}_\Psi$ accepts θ from configuration $\{(\ell_\Phi, J)\}$

$$\text{iff}$$
$$\exists m \geqslant 1 : (\theta, m) \models \varphi_2 \wedge \tau_m \in I - \inf(J) \wedge \tau_m \in I - \sup(J)$$
$$\wedge \forall 1 \leqslant m' < m : (\theta, m') \models \varphi_1.$$

($\Rightarrow$) As automaton $\mathcal{A}_\Psi$ accepts θ from configuration $\{(\ell_\Phi, J)\}$, there exists an accepting run of $\mathcal{A}_\Psi$ on θ from $\{(\ell_\Phi, J)\}$, say

$$\pi = C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \dots \overset{t_i}{\rightsquigarrow} C_{2i-1} \xrightarrow{\sigma_i} C_{2i} \dots,$$

where $C_0 = \{(\ell_\Phi, J)\}$. For all i, when reading σ_i, two transitions can be taken: either $x.\delta(\varphi_2, \sigma) \wedge x \in I$ or $x.\delta(\varphi_1, \sigma) \wedge \varphi_1 U_I \varphi_2 \wedge x \leqslant \sup(I)$. Let m be the first position in the run where the transition $x.\delta(\varphi_2, \sigma) \wedge x \in I$ is taken. Such a position must exist because ℓ_Φ is not an accepting location but π is an accepting run. Then, for all $m' < m$, when reading $\sigma_{m'}$, the transition $x.\delta(\varphi_1, \sigma_j) \wedge \varphi_1 U_I \varphi_2 \wedge x \leqslant \sup(I)$ is taken: it does not reset clock copies that stay in ℓ_Φ. So, the part of configuration C_{2m-1} associated with location ℓ_Φ is $\{(\ell_\Phi, J + \tau_m)\}$. As the transition $x.\delta(\varphi_2, \sigma_m) \wedge x \in I$ is then taken, $J + \tau_m$ must satisfy $x \in I$,

i.e. $\forall v + \tau_m \in J + \tau_m : \ v + \tau_m \in I$ (by definition of the minimal model)
i.e. $\forall v \in J : \ \tau_m \in I - v$

and in particular (as J closed) $\tau_m \in I - \inf(J) \wedge \tau_m \in I - \sup(J)$. Moreover, as π is an accepting run, the part $x.\delta(\varphi_2, \sigma_m)$ of the transition taken from $\{(\ell_\Phi, J + \tau_m)\}$ corresponds to the fact that $C_{2m} \models_{[0,0]} \delta(\varphi_2, \sigma_m)$, thanks to Lemma A.5, we know it means that $(\theta, m) \models \varphi_2$. In the same way, when reading $\sigma_{m'}$, for $1 \leqslant m' < m$, the transition $x.\delta(\varphi_1, \sigma_{m'}) \wedge \varphi_1 U_I \varphi_2 \wedge x \leqslant \sup(I)$ was taken. As π is an accepting run, the part $x.\delta(\varphi_1, \sigma_{m'})$ of the transition taken from $\{(\ell_\Phi, J + \tau_m)\}$ corresponds to the fact that $C_{2m'} \models_{[0,0]} \delta(\varphi_1, \sigma_{m'})$, thanks to Lemma A.5, we know it means that $(\theta, m') \models \varphi_1$. We conclude that $\exists m \geqslant 1 : (\theta, m) \models \varphi_2 \wedge \tau_m \in I - \inf(J) \wedge \tau_m \in I - \sup(J) \wedge \forall 1 \leqslant m' < m : (\theta, m') \models \varphi_1$.

($\Leftarrow$) We will construct an accepting run π of $\mathcal{A}_\Psi$ on θ from configuration $\{(\ell_\Phi, J)\}$, say $C_0 = \{(\ell_\Phi, J)\} \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \dots \overset{t_i}{\rightsquigarrow} C_{2i-1} \xrightarrow{\sigma_i} C_{2i} \dots$. By hypothesis, $\exists m \geqslant 0$ such that (a) $(\theta, m) \models \varphi_2$, (b) $\tau_m \in I - \inf(J) \wedge \tau_m \in$

$I - \sup(J)$ and (c) $\forall 1 \leqslant m' < m : (\theta, m') \models \varphi_1$. From ℓ_Φ we have two possible transitions $x.\delta(\varphi_2, \sigma) \wedge x \in I$ and $x.\delta(\varphi_1, \sigma) \wedge \varphi_1 U_I \varphi_2 \wedge x \leqslant \sup(I)$. We construct π in way it consists of following the transition $x.\delta(\varphi_1, \sigma_{m'}) \wedge \varphi_1 U_I \varphi_2 \wedge x \leqslant \sup(I)$, $\forall 1 \leqslant m' < m$, and the transition $x.\delta(\varphi_2, \sigma_m) \wedge x \in I$ reading σ_m. We must prove that π is an accepting run of $\mathcal{A}_\Psi$ on θ.

Remark that following transition $x.\delta(\varphi_1, \sigma_{m'}) \wedge \varphi_1 U_I \varphi_2 \wedge x \leqslant \sup(I)$, $\forall 1 \leqslant m' < m$, in particular, we loop on ℓ_Φ without reset of clock. It means that, $\forall 1 \leqslant m' < m$, $C_{2m'+1}(\ell_\Phi) = \{J + \tau_{m'}\}$. So, it is possible to take transition $x.\delta(\varphi_1, \sigma_{m'}) \wedge \varphi_1 U_I \varphi_2 \wedge x \leqslant \sup(I)$ reading $\sigma_{m'}$ from ℓ_Φ because the interval associated with this location is then $J + \tau_{m'}$ and satisfies the clock constraint $x \leqslant \sup(I)$: (b) implies that $\tau_{m'} < \tau_m \leqslant \sup(I) - \sup(J)$, so $\sup(J) + \tau_{m'} \leqslant \sup(I)$, and so $\forall j + \tau_{m'} \in J + \tau_{m'}, j + \tau_{m'} \leqslant \sup(I)$. Moreover, we know that $\forall 1 \leqslant m' < m, (\theta, m') \models \varphi_1$. It means that, $\forall 1 \leqslant m' < m$, the automaton $\mathcal{A}_{\varphi_1}$ accepts $\theta^{m'}$ from $\{(\varphi_{1,init}, 0)\}$, i.e. there is an accepting run of $\mathcal{A}_{\varphi_1}$ on $\theta^{m'}$ taking transition $x.\delta(\varphi_1, \sigma_{m'})$ (the unique transition we can take from location $\varphi_{1,init}$). However, the locations of $\mathcal{A}_{\varphi_1}$ in which leads $x.\delta(\varphi_1, \sigma_{m'})$ can be assimilated to the locations of $\mathcal{A}_\Psi$ corresponding to the same formulas (see definitions of such automata and their locations). So, there is also an accepting run of $\mathcal{A}_\Psi$ on $\theta^{m'}$ taking transition $x.\delta(\varphi_1, \sigma_{m'})$ ($\forall 1 \leqslant m' < m$). As transitions $x.\delta(\varphi_1, \sigma_{m'}) \wedge \varphi_1 U_I \varphi_2 \wedge x \leqslant \sup(I)$ loop on ℓ_Φ, when reading σ_m, the interval $J + \tau_m$ is still associated with location ℓ_Φ. π then consists of taking transition $x.\delta(\varphi_2, \sigma_m) \wedge x \in I$. It is possible to take this transition reading σ_m because $J + \tau_m$ satisfies the clock constraint $x \in I$: as $\tau_m \in I - \inf(J) \wedge \tau_m \in I - \sup(J)$ and J is an interval, $\forall j \in J, \tau_m \in I - j$, i.e. $\forall j \in J, j + \tau_m \in I$ and so $\forall v \in J + \tau_m, v \in I$. Moreover, we know that $(\theta, m') \models \varphi_2$. It means that the automaton $\mathcal{A}_{\varphi_2}$ accepts θ^m from $\{(\varphi_{2,init}, 0)\}$, i.e. there is an accepting run of $\mathcal{A}_{\varphi_2}$ on θ^m taking transition $x.\delta(\varphi_2, \sigma_m)$ (the unique transition we can take from location $\varphi_{2,init}$). By the same argument than for φ_1, there is also an accepting run of $\mathcal{A}_\Psi$ on $\theta^{m'}$ taking transition $x.\delta(\varphi_2, \sigma_m)$. We conclude that π is an accepting run of $\mathcal{A}_\Psi$ on θ. □

Proposition 4.9: *Let Φ be an MITL formula, let K be a set of indices and,*

$\forall k \in K$, let $\Phi_k := \Phi_{1,k}\tilde{U}_{I_k}\Phi_{2,k}$ be sub-formulas of Φ. For all $k \in K$, let ℓ_{Φ_k} be their associated locations in $\mathcal{A}_\Phi$. Let $\theta = (\overline{\sigma}, \overline{\tau})$ be an infinite timed word and $J_k \in \mathcal{I}(\mathbb{R}^+)$.

The automaton $\mathcal{A}_\Phi$ accepts θ from configuration $\{(\ell_{\Phi_k}, J_k)_{k\in K}\}$
iff
$\forall k \in K, \forall v \in J_k$, the automaton $\mathcal{A}_\Psi$ accepts θ from configuration $\{(\ell_{\Phi_k}, [v, v])\}$
(i.e.: $\forall k \in K, \forall v \in J_k, (\theta, 1) \models \Phi_{1,k}\tilde{U}_{I_k - v_k}\Phi_{2,k}$).

Proof. Thanks to Lemme A.4, we only need to prove the following. Let Ψ be an MITL formula, let $\Phi := \varphi_1\tilde{U}_I\varphi_2$ be a subformula of Ψ. Let ℓ_Φ be its associated location in $\mathcal{A}_\Psi$. Let $\theta = (\overline{\sigma}, \overline{\tau})$ be an infinite timed word and $J \in \mathcal{I}(\mathbb{R}^+)$. The automaton $\mathcal{A}_\Psi$ accepts θ from configuration $\{(\ell_\Phi, J)\}$ iff $\forall v \in J$, the automaton $\mathcal{A}_\Psi$ accepts θ from configuration $\{(\ell_\Phi, [v, v])\}$ (i.e.: $\forall v \in J, \theta \models \varphi_1\tilde{U}_{I-v}\varphi_2$).
($\Rightarrow$) As $\mathcal{A}_\Psi$ accepts θ from $\{(\ell_\Phi, J)\}$, there is an accepting run π of $\mathcal{A}_\Psi$ on θ from $C_0 = \{(\ell_\Phi, J)\}$, say π:

$$C_0 \overset{t_1}{\rightsquigarrow} C_1 \xrightarrow{\sigma_1} C_2 \overset{t_2}{\rightsquigarrow} C_3 \xrightarrow{\sigma_2} \dots \overset{t_i}{\rightsquigarrow} C_{2i-1} \xrightarrow{\sigma_i} C_{2i} \dots .$$

Let $v \in J$, we will inductively build an accepting run π' of $\mathcal{A}_\Psi$ on θ from $D_0 = \{(\ell_\Phi, [v, v])\}$, say $D_0 \overset{t_1}{\rightsquigarrow} D_1 \xrightarrow{\sigma_1} D_2 \overset{t_2}{\rightsquigarrow} D_3 \xrightarrow{\sigma_2} \dots \overset{t_i}{\rightsquigarrow} D_{2i-1} \xrightarrow{\sigma_i} D_{2i} \dots$ on θ such that the following *invariant* holds: for all $i \geqslant 0$, for all $(\ell, [u, u]) \in D_i$, there is $(\ell, I) \in C_i$ such that $u \in I$. The *base case* is trivial by definition of C_0 and D_0. For the *inductive case*, we first observe that the elapsing of time maintains the invariant. A discrete transition labelled by σ, from some configuration C_{2j+1} in π consists in selecting an arc a_s of the form (ℓ, σ, γ) for each $s = (\ell, I)$ in C_{2j+1}, whose guard is satisfied by I. Then, firing all these arcs yields the configuration C_{2j+2}. From each $s' = (\ell, [v, v])$ in D_{2j+1}, we fire the arc a_s where $s = (\ell, I)$ is a state in C_{2j+1} s.t $v \in I$. Such an s exists by induction hypothesis. Since the effects of the arcs are the same, we conclude that D_{2j+2} and C_{2j+2} respect the invariant. We conclude that π' is accepting thanks to the invariant and the fact that π is accepting.

($\Leftarrow$) We have an accepting run π_v of $\mathcal{A}_\Psi$ on θ from each configuration $\{(\ell_\Phi, v)\}$, say $C_0^v = \{(\ell_\Phi, v)\} \overset{t_1}{\rightsquigarrow} C_1^v \overset{\sigma_1}{\longrightarrow} C_2^v \overset{t_2}{\rightsquigarrow} \ldots \overset{t_i}{\rightsquigarrow} C_{2i-1}^v \overset{\sigma_i}{\longrightarrow} C_{2i}^v \ldots$. We will construct an accepting run π' of $\mathcal{A}_\Psi$ on θ from configuration $\{(\ell_\Phi, J)\}$, say $C_0 = \{(\ell_\Phi, J)\} \overset{t_1}{\rightsquigarrow} C_1 \overset{\sigma_1}{\longrightarrow} C_2 \overset{t_2}{\rightsquigarrow} \ldots \overset{t_i}{\rightsquigarrow} C_{2i-1} \overset{\sigma_i}{\longrightarrow} C_{2i} \ldots$. Remark that the six transitions we can take on this run from ℓ_Φ are: '$x.\delta(\varphi_2, \sigma_i) \wedge x.\delta(\varphi_1, \sigma_i)$', '$x.\delta(\varphi_2, \sigma_i) \wedge \varphi_1 \tilde{U}_I \varphi_2$', '$x.\delta(\varphi_2, \sigma_i) \wedge x > sup(I)$', '$x \notin I \wedge x.\delta(\varphi_1, \sigma_i)$', '$x \notin I \wedge \varphi_1 \tilde{U}_I \varphi_2$' and '$x \notin I \wedge x > sup(I)$'. So, as long as a transition containing '$\varphi_1 \tilde{U}_I \varphi_2$' is taken, the clock copy present in ℓ_Φ is not reset and the part of configurations C_{2i} associated with ℓ_Φ will be $\{(\ell_\Phi, J + \tau_i)\}$ (assuming $\tau_0 = 0$). We distinguish two cases to construct π':

1. <u>if φ_2 is verified on each reading of a letter in $K := \bigcup_{v \in J} I - v$:</u> then π' consists of taking the transition '$x \notin I \wedge \varphi_1 \tilde{U}_I \varphi_2$' on each reading of a letter in an instant $\tau_i < K$. In such instants, the part of configuration associated with ℓ_Φ we are in is $\{(\ell_\Phi, J + \tau_i)\}$ and we indeed satisfy $\forall u \in J + \tau_i, u \notin I$; else $\exists u \in J + \tau_i$ such that $u \in I$ and so $u - \tau_i \in J$ and $u - (u - \tau_i) = \tau_i \in K$, which contradicts our hypothesis.
   Then, π' consists of taking the transition '$x.\delta(\varphi_2, \sigma_i) \wedge \varphi_1 \tilde{U}_I \varphi_2$' on each reading of a letter in an instant $\tau_m \in K$. We know (hypothesis of this case 1.) that in all these instants, $(\theta, m) \models \varphi_2$. It means that the automaton $\mathcal{A}_{\varphi_2}$ accepts θ^m from $\{(\varphi_{1,init}, 0)\}$, i.e. there is an accepting run of $\mathcal{A}_{\varphi_2}$ on θ^m taking transition $x.\delta(\varphi_2, \sigma_m)$ (the unique transition we can take from location $\varphi_{2,init}$). However, the locations of $\mathcal{A}_{\varphi_2}$ in which leads $x.\delta(\varphi_2, \sigma_m)$ can be assimilated to the locations of $\mathcal{A}_\Psi$ corresponding to the same formulas (see definitions of such automata and their locations). So, there is also an accepting run of $\mathcal{A}_\Psi$ on θ^m taking transition $x.\delta(\varphi_2, \sigma_m)$.
   Finally, on the first reading of a letter after K, say in $\tau_j > K$, π' consists of taking the transition '$x \notin I \wedge x > I$'. It is possible because, then, the part of configuration associated with ℓ_Φ we are in is $\{(\ell_\Phi, J + \tau_j)\}$ and $\forall u \in J + \tau_j$: $u > I$. To prove it, suppose that $\exists u \in J + \tau_j : u < I$ or $u \in I$. On the one hand, if $u < I$, as $u \in J + \tau_j$, $\exists v \in J : u = v + \tau_j < I$, i.e.:

$\exists v \in J : \tau_j < I - v$, which contradicts that $\tau_j > K$. On the other hand, if $u \in I$, as $u \in J + \tau_j$, $\exists v \in J : u = v + \tau_j \in I$, i.e.: $\exists v \in J : \tau_j \in I - v$, what contradicts that $\tau_j > K$. π' is so an accepting run of $\mathcal{A}_\Psi$ on θ from configuration $\{(\ell_\Phi, J)\}$.

2. <u>else, φ_1 is verified in a certain instant in $L = \{u' | \exists u \in K : 0 \leqslant u' \leqslant u\}$</u>. Then, there exists a smallest instant $\tau_i \in L$ such that φ_1 is satisfied in τ_i. Moreover, as for each $v \in J, \theta \models \varphi_1 \tilde{U}_{I-v} \varphi_2$, each instant τ_j with $0 \leqslant j \leqslant i$ and $\tau_j \in K$ is an instant in which φ_2 must be satisfied. We must again distinguish two cases:

   - If $\tau_i < K$, then π' consists of taking the transition '$x \notin I \wedge \varphi_1 \tilde{U}_I \varphi_2$' on each reading of a letter in an instant τ_j with $0 \leqslant j < i$ (in such instants, we indeed satisfy $\forall u \in J + \tau_j$, $u \notin I$ because $\tau_j \notin K$) and taking the transition '$x \notin I \wedge x.\delta(\varphi_1, \sigma_i)$' when reading σ_i (it is possible because $\tau_i \notin K$). We can prove that this run is accepting showing, in a similar way as in case 1., that there is an accepting run taking transition $x.\delta(\varphi_1, \sigma_i)$ when reading σ_i.
   - If $\tau_i \in K$, then π' consists of: taking the transition '$x \notin I \wedge \varphi_1 \tilde{U}_I \varphi_2$' on each reading of a letter in an instant τ_j with $0 \leqslant j < i$ and $\tau_j \notin K$ (in such instants, we indeed satisfy $\forall u \in J + \tau_j$, $u \notin I$ because $\tau_j \notin K$) ; taking the transition '$x.\delta(\varphi_2, \sigma_j) \wedge \varphi_1 \tilde{U}_I \varphi_2$' on each reading of a letter in an instant τ_j with $0 \leqslant j < i$ and $\tau_j \in K$ (we know φ_2 is verified in such instants) and taking the transition '$x.\delta(\varphi_2, \sigma_i) \wedge x.\delta(\varphi_1, \sigma_i)$' when reading σ_i (it is possible because as $\tau_i \in K$, φ_2 is satisfied in this instant). We can prove that this run is accepting showing, in a similar way as in case 1., that there is an accepting run taking transitions $x.\delta(\varphi_2, \sigma_j)$ and $x.\delta(\varphi_2, \sigma_i) \wedge x.\delta(\varphi_1, \sigma_i)$ when reading σ_i.

$\square$

APPENDIX B

Proofs of the bisimulation lemma over finite and infinite words

Here is the proof of the bisimulation lemma stated over finite words in Section 4.2.

Proposition 4.23: *Let Φ be an MITL formula, $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$ be a timed automaton and $\mathcal{S}_{\mathcal{B},\neg\Phi}$, of state space S, be the transition system given by Definition 4.17. Let $(B_1, C_1), (B_2, C_2) \in S$ such that $(B_1, C_1) \equiv (B_2, C_2)$. Then:*

1. *for each transition*
   $(B_1, C_1) \overset{t}{\rightsquigarrow} (A_1, D_1)$ *with* $t \in \mathbb{R}^+$ *and* $(A_1, D_1) \in S$,
   there exists $t' \in \mathbb{R}^+$ *and* $(A_2, D_2) \in S$ *such that:*
   $(B_2, C_2) \overset{t'}{\rightsquigarrow} (A_2, D_2)$ *and* $(A_1, D_1) \equiv (A_2, D_2)$;

2. *for each transition*
   $(B_1, C_1) \xrightarrow{\sigma} (A_1, D_1)$, *with* $\sigma \in \Sigma$ *and* $(A_1, D_1) \in S$,
   there exists $(A_2, D_2) \in S$ *such that:*

$$(B_2, C_2) \xrightarrow{\sigma} (A_2, D_2) \quad \textit{and} \quad (A_1, D_1) \equiv (A_2, D_2).$$

Proof. We let $B_1 = (b, v)$ and $C_1 = \{(\ell_k, I_k)_{k \in K}\}$, as $(B_1, C_1) \equiv (B_2, C_2)$, we can let $B_2 = (b, v')$ and $C_2 = \{(\ell_k, I'_k)_{k \in K}\}$.
1. Let us suppose that $(B_1, C_1) \xrightarrow{\sigma} (A_1, D_1)$, with $\sigma \in \Sigma$. Let us note $D_1 = \{(\ell_q, J_q)_{q \in Q}\}$ and $A_1 = (r, v^\star)$. On the one hand, we have that $C_1 \xrightarrow{\sigma} D_1$ and so $D_1 \in f^\star_{\neg\Phi}(E_1)$, where $E_1 = \left(\bigcup_{k \in K} E_k\right)^{\neq}$ and each E_k is a minimal model of $\delta(\ell_k, \sigma)$ with respect to I_k. We recall that $E_1 = \left(\bigcup_{k \in K} a_k[I_k]\right)^{\neq}$ where a_k is a set of atoms as described in Remarks 3.23 and 3.24. Let $E_2 = \left(\bigcup_{k \in K} a_k[I'_k]\right)^{\neq}$, i.e., we use the same arcs from C_1 to E_1 than from C_2 to E_2, for each $D_2 \in f^\star_{\neg\Phi}(E_2)$, we have that $C_2 \xrightarrow{\sigma}_{f^\star_{\neg\Phi}} D_2$ (because $(B_1, C_1) \equiv (B_2, C_2)$ and so, thanks to condition 2. of the definition of '$\equiv$', I_k and I'_k satisfy the same clock constraints). On the other hand, $(B_1, C_1) \xrightarrow{\sigma} (A_1, D_1)$ means that $B_1 \xrightarrow{\sigma} A_1$: there exists an arc $(b, \sigma, r, c, R) \in \delta^{\mathcal{B}}$ such that

$$v \models c \text{ and } \forall x_p \in R : v^\star(x_p) = 0, \text{ while } \forall x_p \notin R : v^\star(x_p) = v(x_p).$$

Let us note $A_2 = (r, v'^\star)$ where $v'^\star$ is such that

$$\forall x_p \in R : v'^\star(x_p) = 0, \text{ while } \forall x_p \notin R : v'^\star(x_p) = v'(x_p).$$

We have that $B_2 \xrightarrow{\sigma} A_2$ following the arc (b, σ, r, c, R): as $(B_1, C_1) \equiv (B_2, C_2)$, the condition 2. of the definition of '$\equiv$' enables $v(x_1), \ldots, v(x_n)$ and $v'(x_1), \ldots, v'(x_n)$ to satisfy the same clock constraints (so that $v' \models c$). We will prove that $(A_1, E_1) \equiv (A_2, E_2)$. The clock values observed to verify conditions 2. to 8. are included in:

$$\begin{aligned} Val \ := \ & \{v(x_p) | 1 \leqslant p \leqslant n\} \cup \{v'(x_p) | 1 \leqslant p \leqslant n\} \cup \{inf(I_k) | k \in K\} \\ & \cup \{sup(I_k) | k \in K\} \cup \{inf(I'_k) | k \in K\} \cup \{sup(I'_k) | k \in K\} \cup \{0\} \end{aligned}$$

(because the discrete transitions either let the clocks values unchanged or replace them by 0). However, conditions 2. to 8. were verified on

$$Val\backslash\{0\}$$

thanks to the fact that $s_1 \equiv s_2$. Moreover if a clock value is replaced by 0 in z or E_1, the corresponding clock value in z' or E_2 is also replaced by 0 (which is the smallest possible clock value, so that its comparisons with all other clocks values will enable to verify 3. to 8.). So, $(A_1, E_1) \equiv (A_2, E_2)$.
Now, as $D_1 \in f^{\star}_{\neg\Phi}(E_1)$, we choose $D_2 \in f^{\star}_{\neg\Phi}(E_2)$ such that the intervals of E_2 grouped to obtain D_2 correspond to those grouped in E_1 to obtain D_1. Let us recall that $D_1 = \{(\ell_q, J_q)_{q\in Q}\}$ and let us note $D_2 = \{(\ell_q, J'_q)_{q\in Q}\}$. Formally, we want D_2 to be such that $\forall(\ell_k, I_k), (\ell_{\tilde{k}}, I_{\tilde{k}}) \in C_1$:

$$\begin{array}{l} \Big((\ell_q, J_q) \in \mathsf{dest}(C_1, D_1, (\ell_k, I_k)) \wedge (\ell_q, J_q) \in \mathsf{dest}(C_1, D_1, (\ell_{\tilde{k}}, I_{\tilde{k}}))\Big) \\ \Rightarrow \quad \Big((\ell_q, J'_q) \in \mathsf{dest}(C_2, D_2, (\ell_k, I'_k)) \wedge (\ell_q, J'_q) \in \mathsf{dest}(C_2, D_2, (\ell_{\tilde{k}}, I'_{\tilde{k}}))\Big). \end{array}$$

We conclude that $(A_1, D_1) \equiv (A_2, D_2)$ (the arguments are the same as those why $(A_1, E_1) \equiv (A_2, E_2)$).
2. Let us suppose that $(B_1, C_1) \overset{t}{\rightsquigarrow} (A_1, D_1)$ for a certain $t \in \mathbb{R}^+$. We must prove there exists $t' \in \mathbb{R}^+$ and a configuration (A_2, D_2) such that

$$(B_2, C_2) \overset{t'}{\rightsquigarrow} (A_2, D_2). \tag{B.1}$$

To do that, we first define the 'time successor' of an element (B, C) of S, noted $next(B, C)$: it is an element of the first equivalence class of $\equiv$ reachable from the class of (B, C) (and different from it) letting time elapsing. Then, we prove that $next(B_1, C_1) \equiv next(B_2, C_2)$, from which we finally deduce B.1.
Let $(B, C) \in S$ with $B = (\bar{b}, \overline{v})$ and $C = \{(\overline{\ell}_k, \overline{I}_k)_{k\in\overline{K}}\}$. Let

$$V \ := \ \{\overline{v}(x_p) | 1 \leqslant p \leqslant n\} \cup \{v | (\ell_k, I_k) \in C \wedge (v = \inf(\overline{I}_k) \vee v = sup(\overline{I}_k))\}$$

be the set of clock values present in B and C. We note $\mu = \max\{frac(v) | v \in V\}$. We define d as $\frac{1-\mu}{2}$ if V contains an integer smaller or equal to c_{max}, and $1 - \mu$

otherwise. We define the time successor of (B, C) to be:

$$next(B, C) \ := \ \left((\overline{b}, \overline{v} + d), \{(\overline{\ell}_k, \overline{I}_k + d)_{k \in \overline{K}}\}\right).$$

We claim that:

$$(B_1, C_1) \equiv (B_2, C_2) \ \text{ implies } \ next(B_1, C_1) \equiv next(B_2, C_2). \tag{B.2}$$

Indeed, we know $next(B_1, C_1) = ((b, v + d), \{(\ell_k, I_k + d)_{k \in K}\})$ for a certain $d > 0$, and $next(B_2, C_2) = ((b, v' + d'), \{(\ell_k, I'_k + d')_{k \in K}\})$ for a certain $d' > 0$. The effect on (B_1, C_1) is either, if an integer is present among the clocks values, to keep the order of their fractional parts unchanged, either, otherwise, to permute the order of the fractional parts of the clocks values with the largest fractional parts such that they now have a zero fractional part (and so are the clock values with the smallest fractional parts). The effect on (B_2, C_2) being the same, in the same cases, and the conditions of $(B_1, C_1) \equiv (B_2, C_2)$ certifying that an integer is present among the clocks values of (B_1, C_1) iff there is an integer between the clocks values of s_2, conditions 1. to 8. are still verified on $next(B_1, C_1)$ and $next(B_2, C_2)$: $next(B_1, C_1) \equiv next(B_2, C_2)$.
Now, as $(B_1, C_1) \xrightarrow{t} (A_1, D_1)$: it means there exists an $n \geqslant 0$ such that $(A_1, D_1) \equiv next^n(B_1, C_1)$. As $(B_1, C_1) \equiv (B_2, C_2)$, we deduce from B.2 that: $next^n(B_1, C_1) \equiv next^n(B_2, C_2)$ and so, taking $(A_2, D_2) = next^n(B_2, C_2)$, we verify B.1 ($d'$ is the sum of the n d's used to recursively computed $next(B_2, C_2), next^2(B_2, C_2), \ldots,$ $next^n(B_2, C_2)$). $\square$

We now present the proof of the bisimulation lemma stated over infinite words in Section 5.4. It is very close to the previously presented proof but takes into account the Miyano-Hayashi markers.

Proposition 5.23: *Let Φ be an MITL formula, $\mathcal{B} = (\Sigma, B, b_0, X, \delta^{\mathcal{B}}, F^{\mathcal{B}})$ be a timed automaton and $\mathcal{S}_{\mathcal{B}, \neg\Phi}$, of state space S, be the transition system given by Definition 5.18. Let $s_1, s_2 \in S$ such that $s_1 \equiv s_2$. Then:*

1. *for each transition*
   $s_1 \overset{t}{\rightsquigarrow} z_1$ *with* $t \in \mathbb{R}^+$ *and* $z_1 \in S$,

there exists $t' \in \mathbb{R}^+$ *and* $z_2 \in S$ *such that:*
$s_2 \overset{t'}{\rightsquigarrow} z_2$ *and* $z_1 \equiv z_2$;

2. *for each transition*
$s_1 \xrightarrow{\sigma} z_1$, *with* $\sigma \in \Sigma$ *and* $z_1 \in S$,
there exists $z_2 \in S$ *such that:*
$z_1 \xrightarrow{\sigma} z_2$ *and* $z_1 \equiv z_2$.

Proof. Let us note $s_1 = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_b)\}$, as $s_1 \equiv s_2$, we can suppose that $s_2 = \{(\ell_k, I'_k, m_k)_{k\in K}\} \cup \{(b, v', m_b)\}$. In the following, we will use the following notations: $C = \{(\ell_k, I_k)_{k\in K}\}$, $C' = \{(\ell_k, I'_k)_{k\in K}\}$, $s = \{(b, v)\}$ and $s' = \{(b, v')\}$.
1. suppose that $s_1 \xrightarrow{\sigma} z_1$, with $\sigma \in \Sigma$ and $z_1 = \{(\ell_q, J_q, m_q)_{q\in Q}\} \cup \{(r, v^\star, m_r)\}$. Let us note $D = \{(\ell_q, J_q)_{q\in Q}\}$ and $z = (r, v^\star)$. On the one hand, we have that $C \xrightarrow{\sigma}_{f^\star_{\neg\Phi}} D$ and so $D \in f^\star_{\neg\Phi}(E)$, where $E = \bigcup_{k\in K} E_k$ and each E_k is a minimal model of $\delta(\ell_k, \sigma)$ with respect to I_k. We recall that $E = \left(\bigcup_{k\in K} a_k[I_k]\right)^{\neq}$ where a_k is a set of atoms as described in Remarks 3.23 and 3.24. Let $E' = \left(\bigcup_{k\in K} a_k[I'_k]\right)^{\neq}$, i.e., we use the same arcs from C to E than from C' to E', for each $D' \in f^\star_{\neg\Phi}(E')$, we have that $C' \xrightarrow{\sigma}_{f^\star_{\neg\Phi}} D'$ (because as $s_1 \equiv s_2$, thanks to condition 2. of the definition of "$\equiv$", I_k and I'_k satisfy the same clock constraints). On the other hand, $s_1 \xrightarrow{\sigma} z_1$ means that $s \xrightarrow{\sigma} z$: there exists an arc $(b, \sigma, c, R, r) \in \delta^{\mathcal{B}}$ such that

$$v \models c \text{ and } \forall x_p \in R : v^\star(x_p) = 0, \text{ while } \forall x_p \notin R : v^\star(x_p) = v(x_p).$$

Let us note $z' = (r, v'^\star)$ where $v'^\star$ is such that

$$\forall x_p \in R : v'^\star(x_p) = 0, \text{ while } \forall x_p \notin R : v'^\star(x_p) = v'(x_p).$$

We have that $s' \xrightarrow{\sigma} z'$ following the arc (b, σ, c, R, r): as $s_1 \equiv s_2$, the condition 2. of the definition of "$\equiv$" enables $v(x_1), \ldots, v(x_n)$ and $v'(x_1), \ldots, v'(x_n)$ to satisfy

the same clock constraints (so that $v' \models c$). Let us note $z_3 = \{(\ell, I, m) | (\ell, I) \in E\} \cup \{(r, v^\star, m_r)\}$ the [1] element of S such that $s \xrightarrow{\sigma} z_3$, and $z_4 = \{(\ell, I, m) | (\ell, I) \in E'\} \cup \{(r, v'^\star, m'_r)\}$ the unique element of S such that $s \xrightarrow{\sigma} z_3$. We will prove that $z_3 \equiv z_4$. Condition 1. of the definition of $\equiv$ is satisfied because C and C' owned the same markers, as well as s and s' (because $s_1 \equiv s_2$), we chose the same minimal models to go from C to E than from C' to E' and the same arc of $\mathcal{B}$ to go from s to z than from s' to z' (these are the unique parameters for the choice of the markers of z_3 and z_4). We still must prove that conditions 2. to 8. are satisfied. The clock values observed to verify conditions 2. to 8. are included in

$$\begin{aligned} Val \ := \ & \{v(x_p) | 1 \leqslant p \leqslant n\} \cup \{v'(x_p) | 1 \leqslant p \leqslant n\} \cup \{inf(I_k) | k \in K\} \\ & \cup \{sup(I_k) | k \in K\} \cup \{inf(I'_k) | k \in K\} \cup \{sup(I'_k) | k \in K\} \cup \{0\} \end{aligned}$$

(as the discrete transitions either let the clocks values unchanged or replace them by 0). However, conditions 2. to 8. were verified on

$$Val \backslash \{0\}$$

thanks to the fact that $s_1 \equiv s_2$. Moreover if a clock value is replaced by 0 in z or E, the corresponding clock value in z' or E' is also replaced by 0 (which is the smallest possible clock value, so that its comparisons with all other clocks values will enable to verify 3. to 8.). So, $z_3 \equiv z_4$.

Now, as $D \in f^\star_{\neg\Phi}(E)$, we choose $D' \in f^\star_{\neg\Phi}(E')$ such that the intervals of E' grouped to obtain D' correspond to those grouped in E to obtain D. Let us recall that $D = \{(\ell_q, J_q)_{q \in Q}\}$ and let us note $D' = \{(\ell_q, J'_q)_{q \in Q}\}$. Formally, we want D' to be such that: $\forall (\ell_k, I_k), (\ell_{\tilde{k}}, I_{\tilde{k}}) \in C$:

$$\begin{aligned} & \Big((\ell_q, J_q) \in \mathsf{dest}(C, D, (\ell_k, I_k)) \wedge (\ell_q, J_q) \in \mathsf{dest}(C, D, (\ell_{\tilde{k}}, I_{\tilde{k}})) \Big) \\ \Rightarrow \ & \Big((\ell_q, J'_q) \in \mathsf{dest}(C', D', (\ell_k, I'_k)) \wedge (\ell_q, J'_q) \in \mathsf{dest}(C', D', (\ell_{\tilde{k}}, I'_{\tilde{k}})) \Big). \end{aligned}$$

We conclude that $z_1 \equiv z_2$ (the arguments are the same as those why $z_3 \equiv z_4$).

2. Let us suppose that $s_1 \overset{t}{\rightsquigarrow} z_1$ for a certain $t \in \mathbb{R}^+$. We must prove there exists

[1] once the minimal models and the arc of $\mathcal{B}$ of the definition of $\xrightarrow{\sigma}$ are chosen, it is easy to see there exists a unique possible choice for the values of the markers of z_3.

$t' \in \mathbb{R}^+$ and a configuration z_2 such that:

$$s_2 \overset{t'}{\rightsquigarrow} z_2. \tag{B.3}$$

To do that, we first define the "time successor" of an element $\overline{s}$ of S, noted $next(\overline{s})$: it is an element of the first equivalence class of $\equiv$ reachable from the class of s_1 (and different from it) letting time elapsing. Then, we prove that $next(s_1) \equiv next(s_2)$, from which we finally deduce B.3.
Let $\overline{s} \in S$ with $\overline{s} = \{(\overline{\ell}_k, \overline{I}_k, \overline{m}_k)_{k \in \overline{K}}\} \cup \{(\overline{b}, \overline{v}, \overline{m}_b)\}$. Let us note $C = \{(\overline{\ell}_k, \overline{I}_k)_{k \in \overline{K}}\}$ and $s = \{(\overline{b}, \overline{v})\}$. Let

$$V \ := \ \{\overline{v}(x_p) | 1 \leqslant p \leqslant n\} \cup \{v | (\overline{\ell}_k, \overline{I}_k) \in C \wedge (v = \inf(I_k) \vee v = sup(I_k))\}$$

be the set of clock values present in s and C. We note $\mu = \max\{frac(v) | v \in V\}$. We define d as $\frac{1-\mu}{2}$ if V contains an integer smaller or equal to c_{max}, and $1 - \mu$ otherwise. We define the time successor of $\overline{s}$ to be

$$next(B, C) \ := \ \{(\overline{\ell}_k, \overline{I}_k + d, \overline{m}_k)_{k \in \overline{K}}\} \cup \{(\overline{b}, \overline{v} + d, \overline{m}_b)\}.$$

We claim that:

$$s_1 \equiv s_2 \ \text{ implies } \ next(s_1) \equiv next(s_2). \tag{B.4}$$

We know $next(s_1) = \{(\ell_k, I_k + d, m_k)_{k \in K}\} \cup \{(b, v + d, m_b)\}$ for a certain $d > 0$, and $next(s_2) = \{(\ell_k, I'_k + d', m_k)_{k \in K}\} \cup \{(b, v' + d', m_b)\}$ for a certain $d' > 0$. The effect on s_1 is either, if an integer is present among the clocks values, to keep the order of their fractional parts unchanged, either, otherwise, to permute the order of the fractional parts of the clocks values with the largest fractional parts such that they now have a zero fractional part (and so are the clock values with the smallest fractional parts). The effect on s_2 being the same, in the same cases, and the conditions of $s_1 \equiv s_2$ certifying that an integer is present among the clocks values of s_1 iff there is an integer between the clocks values of s_2, conditions 1. to 8. are still verified on $next(s_1)$ and $next(s_2)$: $next(s_1) \equiv next(s_2)$.
Now, as $s_1 \overset{t}{\rightsquigarrow} z_1$: it means there exists an $n \geqslant 0$ such that $z_1 \equiv next^n(s_1)$. As $s_1 \equiv s_2$, we deduce from B.4 that: $next^n(s_1) \equiv next^n(s_2)$ and so, taking $z_2 = next^n(s_2)$, we verify B.3 (d' is the sum of the n d's used to recursively computed $next(s_2), next^2(s_2), \ldots, next^n(s_2)$). $\square$

APPENDIX C

Proof of Proposition 5.20

This section is dedicated to the proof of Proposition 5.20 which is closed to that of Proposition 5.13

Proposition 5.20: *For every MITL formula Φ, the associated $\mathcal{A}_{\neg\Phi}$ and $f^{\star}_{\neg\Phi}$, and for every Büchi timed automaton $\mathcal{B}$:*

$$L^{\omega}(\mathcal{S}_{\mathcal{B},\neg\Phi}) = L^{\omega}_{f^{\star}_{\neg\Phi}}(\mathcal{A}_{\neg\Phi}) \cap L(\mathcal{B}).$$

Proof. ($\supseteq$) Let $\theta = (\overline{\sigma}, \overline{\tau}) \in L^{\omega}_{f^{\star}_{\neg\Phi}}(\mathcal{A}_{\neg\Phi}) \cap L(\mathcal{B})$, with $\overline{\sigma} = \sigma_1\sigma_2\cdots\sigma_n\ldots$ and $\overline{\tau} = \tau_1\tau_2\cdots\tau_n\ldots$. We will prove that $\theta \in L^{\omega}(\mathcal{S}_{\mathcal{B},\neg\Phi})$. Let us note $t_i = \tau_i - \tau_{i-1}$ for all $1 \leqslant i \leqslant |\theta|$, assuming $\tau_0 = 0$. We have an accepting $f^{\star}_{\neg\Phi}$-run of $\mathcal{A}_{\neg\Phi}$ on θ, say $\pi^{\mathcal{A}}$:

$$C^{\mathcal{A}}_0 \overset{t_1}{\rightsquigarrow} C^{\mathcal{A}}_1 \xrightarrow{\sigma_1}_{f^{\star}_{\neg\Phi}} C^{\mathcal{A}}_2 \overset{t_2}{\rightsquigarrow} C^{\mathcal{A}}_3 \xrightarrow{\sigma_2}_{f^{\star}_{\neg\Phi}} \ldots \overset{t_i}{\rightsquigarrow} C^{\mathcal{A}}_{2i-1} \xrightarrow{\sigma_i}_{f^{\star}_{\neg\Phi}} C^{\mathcal{A}}_{2i} \ldots.$$

We also have an accepting run of $\mathcal{B}$ on θ, say $\pi^{\mathcal{B}}$:

$$C^{\mathcal{B}}_0 \overset{t_1}{\rightsquigarrow} C^{\mathcal{B}}_1 \xrightarrow{\sigma_1} C^{\mathcal{B}}_2 \overset{t_2}{\rightsquigarrow} C^{\mathcal{B}}_3 \xrightarrow{\sigma_2} \ldots \overset{t_i}{\rightsquigarrow} C^{\mathcal{B}}_{2i-1} \xrightarrow{\sigma_i} C^{\mathcal{B}}_{2i} \ldots.$$

We must prove that there is an accepting run of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ on θ, say π':

$$s_0 \overset{t_1}{\rightsquigarrow} s_1 \xrightarrow{\sigma_1} s_2 \overset{t_2}{\rightsquigarrow} s_3 \xrightarrow{\sigma_2} \dots \overset{t_i}{\rightsquigarrow} s_{2i-1} \xrightarrow{\sigma_i} s_i \dots .$$

We construct π' by induction, proving additionally that the two following properties hold for $j \geqslant 0$:

($\star 2j$) $C^{\mathcal{A}}_{2j} = \{(\ell_k, I_k)_{k\in K}\}$ and $C^{\mathcal{B}}_{2j} =\} \cup \{(b, v)\}$ iff $s_{2j} = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_{\mathcal{B}})\}$ for some $m_k \in \{\top, \bot\}$, for all $k \in K$, and some $m_{\mathcal{B}} \in \{\top, \bot\}$;

($*2j$) if (i) a location of F occurs on all the branches of $\pi^{\mathcal{A}}$ between the last configuration $C^{\mathcal{A}}_{2j'}$ such that $s_{2j'} \in \alpha$ (or, failing that, between $C^{\mathcal{A}}_0$) and $C^{\mathcal{A}}_{2j}$, and (ii) a location of $F^{\mathcal{B}}$ occurs in $\pi^{\mathcal{B}}$ between the last configuration $C^{\mathcal{B}}_{2j'}$ such that $s_{2j'} \in \alpha$ (or, failing that, between $C^{\mathcal{B}}_0$) and $C^{\mathcal{B}}_{2j}$, then, $s_{2j} \in \alpha$.

<u>Basis:</u> We know that $C^{\mathcal{A}}_0 = \{(\ell_0, [0, 0])\}$, $C^{\mathcal{B}}_0 = \{(b_0, v_0)\}$ and $s_0 = \{(\ell_0, [0, 0], \bot), (b_0, v_0, m)\}$, where v_0 is the valuation such that $v_0(x) = 0$, $\forall x \in X$, and $m = \top$ iff $b_0 \in F^{\mathcal{B}}$. It is clear that ($\star 0$) and ($*0$) are verified because only $\ell_0 \notin F$ occurs on the (unique) branch of π.

<u>Induction:</u> Suppose that we constructed π' until s_{2i} and that ($\star 2j$) and ($*2j$) are verified $\forall 0 \leqslant j \leqslant i$. We will construct π' until $s_{2(i+1)}$ in way ($\star 2(i + 1)$) and ($*2(i + 1)$) will still be verified.

First, we must construct s_{2i+1} such that $s_{2i} \overset{t_{i+1}}{\rightsquigarrow} s_{2i+1}$. Suppose $s_{2i} = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_{\mathcal{B}})\}$, as ($\star 2i$) is verified by hypothesis, it means that $C^{\mathcal{A}}_{2i}$ can be written as $\{(\ell_k, I_k)_{k\in K}\}$ and that $C^{\mathcal{B}}_{2i}$ can be written as $\{(b, v)\}$. We must choose s_{2i+1} to be $\{(\ell_k, I_k + t_{i+1}, m_k)_{k\in K}\} \cup \{(b, v + t_{i+1}, m_{\mathcal{B}})\}$. As $C^{\mathcal{A}}_{2i} \overset{t_{i+1}}{\rightsquigarrow} C^{\mathcal{A}}_{2i+1}$, $C^{\mathcal{A}}_{2i+1} = C^{\mathcal{A}}_{2i} + t_{i+1} = \{(\ell_k, I_k + t_{i+1})_{k\in K}\}$. In a similar way, as $C^{\mathcal{B}}_{2i} \overset{t_{i+1}}{\rightsquigarrow} C^{\mathcal{B}}_{2i+1}$, $C^{\mathcal{B}}_{2i+1} = C^{\mathcal{B}}_{2i} + t_{i+1} = \{(b, v + t_{i+1})\}$. Remark that the following property holds:

($\star 2i + 1$) $C^{\mathcal{A}}_{2i+1} = \{(\ell_k, I_k)_{k\in K}\}$ and $C^{\mathcal{B}}_{2i+1} = \{(b, v)\}$ iff

$s_{2i+1} = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_{\mathcal{B}})\}$ for some $m_k \in \{\top, \bot\}$, for all $k \in K$, and some $m_{\mathcal{B}} \in \{\top, \bot\}$.

Secondly, we must construct $s_{2(i+1)}$ such that $s_{2i+1} \xrightarrow{\sigma_{i+1}} s_{2(i+1)}$. We know that $C^{\mathcal{A}}_{2i+1} \xrightarrow{\sigma_{i+1}}_{f^\star_{\neg\Phi}} C^{\mathcal{A}}_{2(i+1)}$ and $C^{\mathcal{B}}_{2i+1} \xrightarrow{\sigma_{i+1}} C^{\mathcal{B}}_{2(i+1)}$. Suppose $s_{2i+1} = \{(\ell_k, I_k, m_k)_{k\in K}\}$ $\cup\{(b, v, m_{\mathcal{B}})\}$, as $(\star 2i+1)$ is verified, $C^{\mathcal{A}}_{2i+1}$ can be written as $\{(\ell_k, I_k)_{k\in K}\}$ and $C^{\mathcal{B}}_{2i+1}$ as $\{(b, v)\}$. Let us further suppose that $C^{\mathcal{A}}_{2(i+1)} = \{(\ell_{k'}, I_{k'})_{k'\in K'}\}$ and $C^{\mathcal{B}}_{2(i+1)} = \{(b', v')\}$. We construct:

$$s_{2(i+1)} := \{(\ell_{k'}, I_{k'}, m_{k'})_{k'\in K'}\} \cup \{(b', v', m'_{\mathcal{B}})\}$$

to be the unique state[1] of S such that $\{(\ell_k, I_k)_{k\in K}\} \xrightarrow{\sigma_{i+1}}_{f^\star_{\neg\Phi}} \{(\ell_{k'}, I_{k'})_{k'\in K'}\}$ and $(b, v) \xrightarrow{\sigma_{i+1}} (b', v')$. $(\star 2(i+1))$ is trivially verified. It remains to prove that $(*2(i+1))$ is satisfied. Suppose that (i) a location of F occurs on all the branches of $\pi^{\mathcal{A}}$ between the last configuration $C^{\mathcal{A}}_{2j'}$ such that $s_{2j'} \in \alpha$ (or, failing that, between $C^{\mathcal{A}}_0$) and $C^{\mathcal{A}}_{2(i+1)}$, and that (ii) a location of $F^{\mathcal{B}}$ occurs in $\pi^{\mathcal{B}}$ between the last configuration $C^{\mathcal{B}}_{2j'}$ such that $s_{2j'} \in \alpha$ (or, failing that, between $C^{\mathcal{B}}_0$) and $C^{\mathcal{B}}_{2(i+1)}$. We must prove that $s_{2(i+1)} \in \alpha$, i.e. all the trios of $s_{2(i+1)}$ has $\top$ as last component. On the one hand, remark that the hypothesis implies there exists a transition $\xrightarrow{\sigma_j}$ in $\pi^{\mathcal{B}}$, for some $j' \leqslant j \leqslant i+1$, such that $C^{\mathcal{B}}_{2j} = (b, v)$ for some $b \in F^{\mathcal{B}}$. On the second hand, let $\beta = \beta_0\beta_1\beta_2 \ldots \beta_{2j'} \ldots \beta_{2j} \ldots \beta_{2(i+1)} \ldots$ be a branch of $\pi^{\mathcal{A}}$. The hypothesis implies there exists a transition $\xrightarrow{\sigma_j}_{f^\star_{\neg\Phi}}$ of $\pi^{\mathcal{A}}$, for some $j' \leqslant j \leqslant i+1$, such that $\beta_{2j} = (\ell, I)$ for some $\ell \in F$. Thanks to $(*2j)$, $(\ell, I, m_k) \in s_{2j}$, for some m_k in $\{\top, \bot\}$, but point (v)-(b) of the definition of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ (Definition 5.18) obliges m_k to be $\top$. We conclude from these two facts that the third components associated in π' to the different states of the branches of π will gradually (between $s_{2j'}$ and $s_{2(i+1)}$) become $\top$. We must still ensure they will eventually never become $\bot$ again. One the one hand, a pair $(b, v) = C^{\mathcal{B}}_j$ (for $2j' \leqslant j \leqslant 2(i+1)$) of $\pi^{\mathcal{B}}$, corresponding to $(b, v, \top)$ in π', can have a successor (b', v') in $\pi^{\mathcal{B}}$ and a corresponding successor $(b', v', \bot)$ in π' *iff* s_j is accepting and $b' \notin F^{\mathcal{B}}$: this is not possible under the present

[1] once the minimal models of the definition of $\xrightarrow{\sigma_{i+1}}$ are chosen, it is easy to see there exists a *unique* possible choice for the values of the m_k and of $m_{\mathcal{B}}$ of $s_{2(i+1)}$.

hypothesis. On the other hand, a pair $(\ell, I) \in C_j^{\mathcal{A}}$ (for $2j' \leqslant j \leqslant 2(i+1)$) of $\pi^{\mathcal{A}}$, corresponding to $(\ell, I, \top)$ in π', can have a successor (ℓ', I') in $\pi^{\mathcal{A}}$ and a corresponding successor $(\ell', I', \bot)$ in π' *iff* s_j is accepting and $\ell' \notin F$ (which is not possible under the present hypothesis) or $(\ell', I', \bot)$ comes from the grouping of trios (thanks to $f^{\star}_{\neg\Phi}$) emanating from trios such that at least one of them had $\bot$ as last component (case (v)-(c) of the definition of $\mathcal{S}_{\mathcal{B},\neg\Phi}$). It means that no location of F occurs on one of the branches of $\pi^{\mathcal{A}}$ leading to (ℓ', I'), say $\beta' = \beta'_0\beta'_1\beta'_2 \ldots \beta_{2j'} \ldots \beta'_{2k'} \ldots \beta'_{2(i+1)} \ldots$, with $\beta'_{2k} = (\ell', I')$, since $\beta'_{2j'}$ (else, we contradict case (v)-(c) of the definition of $\mathcal{S}_{\mathcal{B},\neg\Phi}$). But, by hypothesis, a location of F occurs on all the branches of $\pi^{\mathcal{A}}$ between steps $2j'$ and $2(i+1)$, so there exists a transition $\xrightarrow{\sigma_{2\tilde{j}}}_{f^{\star}_{\neg\Phi}}$, for $k' < \tilde{j} \leqslant i+1$, such that $\beta'_{2\tilde{j}}$ has its location in F: once again, point (v)-(b) of the definition of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ obliges $m_{2\tilde{j}}$ to be $\top$.
As a location of F occurs on all branches of $\pi^{\mathcal{A}}$ between steps $2j'$ and $2(i+1)$ and there is only a finite number of branches leading to a state of $C^{\mathcal{A}}_{2(i+1)}$, we conclude that we can only encounter this last case a finite number of times and so $s_{2(i+1)} \in \alpha$: $(*2(i+1))$ is satisfied.
To end this part of the proof, we must show that π' is accepting. The previous induction proves that $(*2j)$ is verified for all $j \geqslant 0$. As $\pi^{\mathcal{A}}$ and $\pi^{\mathcal{B}}$ are accepting, we have that: (i) a location of F occurs on all the branches of $\pi^{\mathcal{A}}$ infinitely often, and (ii) a location of $F^{\mathcal{B}}$ occurs infinitely often along $\pi^{\mathcal{B}}$. Hence, there is an infinite number of j and j' such that the antecedent of $(*2j)$ is true. So, in this same infinite number of times, we know that $s_{2j} \in \alpha$, what proves that π' is accepting.
$(\subseteq)$ Let $\theta = (\overline{\sigma}, \overline{\tau}) \in L^{\omega}(\mathcal{S}_{\mathcal{B},\neg\Phi})$, with $\overline{\sigma} = \sigma_1\sigma_2 \cdots \sigma_n \ldots$ and $\overline{\tau} = \tau_1\tau_2 \cdots \tau_n \ldots$.
We will prove that $\theta \in L^{\omega}_{f^{\star}_{\neg\Phi}}(\mathcal{A}_{\neg\Phi}) \cap L(\mathcal{B})$. Let us note $t_i = \tau_i - \tau_{i-1}$ for all $1 \leqslant i \leqslant |\theta|$, assuming $\tau_0 = 0$. We have an accepting run of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ on θ, say π:

$$s_0 \overset{t_1}{\rightsquigarrow} s_1 \xrightarrow{\sigma_1} s_2 \overset{t_2}{\rightsquigarrow} s_3 \xrightarrow{\sigma_2} \ldots \overset{t_i}{\rightsquigarrow} s_{2i-1} \xrightarrow{\sigma_i} s_i \ldots.$$

We must prove that (i) there is an accepting $f^{\star}_{\neg\Phi}$-run of $\mathcal{A}_{\neg\Phi}$ on θ, say $\pi^{\mathcal{A}}$:

$$C_0^{\mathcal{A}} \overset{t_1}{\rightsquigarrow} C_1^{\mathcal{A}} \xrightarrow{\sigma_1}_{f^{\star}_{\neg\Phi}} C_2^{\mathcal{A}} \overset{t_2}{\rightsquigarrow} C_3^{\mathcal{A}} \xrightarrow{\sigma_2}_{f^{\star}_{\neg\Phi}} \ldots \overset{t_i}{\rightsquigarrow} C_{2i-1}^{\mathcal{A}} \xrightarrow{\sigma_i}_{f^{\star}_{\neg\Phi}} C_{2i}^{\mathcal{A}} \ldots.$$

and (ii) there is an accepting run of $\mathcal{B}$ on θ, say $\pi^{\mathcal{B}}$:

$$C_0^{\mathcal{B}} \overset{t_1}{\rightsquigarrow} C_1^{\mathcal{B}} \xrightarrow{\sigma_1} C_2^{\mathcal{B}} \overset{t_2}{\rightsquigarrow} C_3^{\mathcal{B}} \xrightarrow{\sigma_2} \dots \overset{t_i}{\rightsquigarrow} C_{2i-1}^{\mathcal{B}} \xrightarrow{\sigma_i} C_{2i}^{\mathcal{B}} \dots .$$

We construct $\pi^{\mathcal{A}}$ and $\pi^{\mathcal{B}}$ by induction, proving additionally that the two following properties hold for $j \geqslant 0$:

$(\star 2j)$ $C_{2j}^{\mathcal{A}} = \{(\ell_k, I_k)_{k\in K}\}$ and $C_{2j}^{\mathcal{B}} =\} \cup \{(b, v)\}$ iff $s_{2j} = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_{\mathcal{B}})\}$ for some $m_k \in \{\top, \bot\}$, for all $k \in K$, and some $m_{\mathcal{B}} \in \{\top, \bot\}$;

$(*2j)$ if $s_{2j} \in \alpha$, then, (i) a location of F occurs on all the branches of $\pi^{\mathcal{A}}$ between the last configuration $C_{2j'}^{\mathcal{A}}$ such that $s_{2j'} \in \alpha$ (or, failing that, between $C_0^{\mathcal{A}}$) and $C_{2j}^{\mathcal{A}}$, and (ii) a location of $F^{\mathcal{B}}$ occurs in $\pi^{\mathcal{B}}$ between the last configuration $C_{2j'}^{\mathcal{B}}$ such that $s_{2j'} \in \alpha$ (or, failing that, between $C_0^{\mathcal{B}}$) and $C_{2j}^{\mathcal{B}}$.

<u>Basis:</u> We know that $s_0 = \{(\ell_0, [0, 0], \bot), (b_0, v_0, m)\}$, $C_0^{\mathcal{A}} = \{(\ell_0, [0, 0])\}$,and $C_0^{\mathcal{B}} = \{(b_0, v_0)\}$, where v_0 is the valuation such that $v_0(x) = 0$, $\forall x \in X$, and $m = \top$. $(\star 0)$ and $(*0)$ are trivially verified.
<u>Induction:</u> Suppose that we constructed $\pi^{\mathcal{A}}$ and $\pi^{\mathcal{B}}$ until $C_{2i}^{\mathcal{A}}$ and $C_{2i}^{\mathcal{B}}$ and that moreover $(\star 2j)$ and $(*2j)$ are verified $\forall 0 \leqslant j \leqslant i$. We will construct $\pi^{\mathcal{A}}$ and $\pi^{\mathcal{B}}$ until $C_{2(i+1)}^{\mathcal{A}}$ and $C_{2(i+1)}^{\mathcal{B}}$ in way $(\star 2(i + 1))$ and $(*2(i + 1))$ will still hold.
First, we must construct $C_{2i+1}^{\mathcal{A}}$ such that $C_{2i}^{\mathcal{A}} \overset{t_{i+1}}{\rightsquigarrow} C_{2i+1}^{\mathcal{A}}$ and $C_{2i+1}^{\mathcal{B}}$ such that $C_{2i}^{\mathcal{B}} \overset{t_{i+1}}{\rightsquigarrow} C_{2i+1}^{\mathcal{B}}$. We know that $s_{2i} \overset{t_{i+1}}{\rightsquigarrow} s_{2i+1}$. Suppose $s_{2i} = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_{\mathcal{B}})\}$, as $(\star 2i)$ is verified by hypothesis, $C_{2i}^{\mathcal{A}} = \{(\ell_k, I_k)_{k\in K}\}$ and $C_{2i}^{\mathcal{B}} = \{(b, v)\}$. We must choose $C_{2i+1}^{\mathcal{A}}$ to be $\{(\ell_k, I_k + t_{i+1})_{k\in K}\}$ and $C_{2i+1}^{\mathcal{B}}$ to be $\{(b, v + t_{i+1})\}$. As $s_{2i} \overset{t_{i+1}}{\rightsquigarrow} s_{2i+1}$, $s_{2i+1} = \{(\ell_k, I_k + t_{i+1}, m_k)_{k\in K}\} \cup \{(b, v + t_{i+1}, m_{\mathcal{B}})\}$. Remark that the following property holds:

$(\star 2i + 1)$ $C_{2i+1}^{\mathcal{A}} = \{(\ell_k, I_k)_{k\in K}\}$ and $C_{2i+1}^{\mathcal{B}} = \{(b, v)\}$ iff $s_{2i+1} = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_{\mathcal{B}})\}$ for some $m_k \in \{\top, \bot\}$, for all $k \in K$, and some $m_{\mathcal{B}} \in \{\top, \bot\}$.

Secondly, we must construct $C^{\mathcal{A}}_{2(i+1)}$ such that $C^{\mathcal{A}}_{2i+1} \xrightarrow{\sigma_{i+1}}_{f^{\star}_{\neg\Phi}} C^{\mathcal{A}}_{2(i+1)}$ and $C^{\mathcal{B}}_{2(i+1)}$ such that $C^{\mathcal{B}}_{2i+1} \xrightarrow{\sigma_{i+1}} C^{\mathcal{B}}_{2(i+1)}$. We know that $s_{2i+1} \xrightarrow{\sigma_{i+1}} s_{2(i+1)}$. Suppose $s_{2i+1} = \{(\ell_k, I_k, m_k)_{k\in K}\} \cup \{(b, v, m_{\mathcal{B}})\}$, as $(\star 2i+1)$ is verified, $C^{\mathcal{A}}_{2i+1} = \{(\ell_k, I_k)_{k\in K}\}$ and $C^{\mathcal{B}}_{2i+1} = \{(b, v)\}$. Let us further suppose that $s_{2(i+1)} = \{(\ell_{k'}, I_{k'}, m_{k'})_{k'\in K'}\} \cup \{(b', v', m'_{\mathcal{B}})\}$. We construct $C^{\mathcal{A}}_{2(i+1)} = \{(\ell_{k'}, I_{k'})_{k'\in K'}\}$ and $C^{\mathcal{B}}_{2(i+1)} = \{(b', v')\}$, so that $(\star 2(i+1))$ holds. Remark that, by definition of $\rightarrow$: $s_{2i+1} \xrightarrow{\sigma_{i+1}} s_{2(i+1)}$ and so we have $C^{\mathcal{A}}_{2i+1} \xrightarrow{\sigma_{i+1}}_{f^{\star}_{\neg\Phi}} C^{\mathcal{A}}_{2(i+1)}$ and $C^{\mathcal{B}}_{2i+1} \xrightarrow{\sigma_{i+1}} C^{\mathcal{B}}_{2(i+1)}$ (what we needed). It remains to prove that $(*2(i+1))$ is satisfied. Suppose that $s_{2(i+1)} \in \alpha$ (i.e. $\forall k' \in K'$, $m_{k'} = \top$). We must prove that (i) between the last configuration $C^{\mathcal{A}}_{2j'}$ such that $s_{2j'} \in \alpha$ (or, failing that, between $C^{\mathcal{A}}_0$) and $C^{\mathcal{A}}_{2(i+1)}$, a location of F occurs on all the branches of $\pi^{\mathcal{A}}$, and that (ii) a location of $F^{\mathcal{B}}$ occurs in $\pi^{\mathcal{B}}$ between the last configuration $C^{\mathcal{B}}_{2j'}$ such that $s_{2j'} \in \alpha$ (or, failing that, between $C^{\mathcal{B}}_0$) and $C^{\mathcal{B}}_{2(i+1)}$. Let us prove these two facts by contradiction. On the one hand, suppose that no location of $F^{\mathcal{B}}$ occurs in $\pi^{\mathcal{B}}$ between the last configuration $C^{\mathcal{B}}_{2j'}$ such that $s_{2j'} \in \alpha$ (or, failing that, between $C^{\mathcal{B}}_0$) and $C^{\mathcal{B}}_{2(i+1)}$. Then, as $s_{2j'} \in \alpha$, all the third components of $s_{2j'}$ are replaced by $\perp$ (likewise, by the previous hypothesis and by definition of s_0, its unique trio whose location is in B has $\perp$ as last component) before evolving reading $\sigma_{j'+1}, \sigma_{j'+2}, \ldots, \sigma_{i+1}$ thanks to the rules in (v) in the definition of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ (Definition 5.18). Hence, the trio of $s_{2j'}$ corresponding to $C^{\mathcal{B}}_{2j'}$ can only evolve to a trio with $\top$ as last component if case (d) is satisfied, what is impossible between $C^{\mathcal{B}}_{2j'}$ and $C^{\mathcal{B}}_{2(i+1)}$. This contradicts the fact that $s_{2(i+1)} \in \alpha$. On the second hand, let us consider a branch $\beta = \beta_0\beta_1\beta_2 \ldots \beta_{2j'} \ldots \beta_{2j} \ldots \beta_{2(i+1)} \ldots$ of $\pi^{\mathcal{A}}$ and suppose that $\forall j' \leqslant j \leqslant i+1$, β_{2j} has not its location in F. As $s_{2j'} \in \alpha$, all the third components of $s_{2j'}$ are replaced by $\perp$ (likewise, by definition of s_0, its unique trio whose location is in L has $\perp$ as last component) before evolving reading $\sigma_{j'+1}, \sigma_{j'+2}, \ldots, \sigma_{i+1}$ thanks to the rules in (v) in the definition of $\mathcal{S}_{\mathcal{B},\neg\Phi}$ (Definition 5.18). But, observing those rules in (v), when a trio has $\perp$ as last component, it can only evolve to a trio with $\top$ as last component if the 'otherwise' part of case (c) is satisfied, what is impossible along β. This contradicts the fact that $s_{2(i+1)} \in \alpha$.

To end the proof, we must show that $\pi^{\mathcal{A}}$ and $\pi^{\mathcal{B}}$ are accepting. The previous induction proves that $(*2j)$ holds for all $j \geqslant 0$. As π is accepting, we know that $s_{2j} \in \alpha$ for infinitely many j and so, between any two successive such j's all the branches of $\pi^{\mathcal{A}}$ visit F and $\pi^{\mathcal{B}}$ visits $F^{\mathcal{B}}$. We conclude that (i) all the branches of $\pi^{\mathcal{A}}$ visit F infinitely often, what proves that $\pi^{\mathcal{A}}$ is accepting, and that (ii) $\pi^{\mathcal{B}}$ visit $F^{\mathcal{B}}$ infinitely often, what proves that $\pi^{\mathcal{B}}$ is accepting. $\square$

APPENDIX D

Other proofs of Section 5.4

One can find in this section the proofs of the propositions stated in Section 5.4.

Proposition 5.27: *Let $s, s' \in S$. We have: $s \equiv s'$ iff $H(s) = H(s')$.*

Proof. ($\Rightarrow$) Suppose that $s \equiv s'$, then the order of the fractional parts of all the clock values that s contains is the same than those of all the corresponding clock values that s' contains (see $\equiv$ conditions 3. to 8.). Moreover, their corresponding states have their infima (respectively their suprema, respectively clocks values) in the same region (see $\equiv$ condition 2.) and their locations are the same. The way $H(s)$ and $H(s')$ are constructed, their will be no difference between these two words.
($\Leftarrow$) Suppose $H(s) = H(s')$, then we associate each interval of the configuration s of $\mathcal{A}_{\neg\Phi}$, clearly defined by two 4-tuples in $H(s)$, to the interval of s' that is represented by the two corresponding 4-tuples of $H(s')$. Moreover, for $1 \leqslant i \leqslant n$, we associate each value v_i of the clock x_i of $\mathcal{B}$, clearly defined by a certain 4-tuple in $H(s)$, with the value of v'_i of the clock x_i of $\mathcal{B}$ represented in the corresponding 4-tuple of $H(s')$. As $H(s) = H(s')$, conditions 1. and 2. of $\equiv$ are of course verified.

The other conditions are also respected thanks to the groupings executed on the elements of $H(s)$ and $H(s')$ to reflect the increasing order of the fractional parts of the second components (i.e. clock) values. □

Proposition 5.30: *Let $W^1, W^2 \in \mathcal{H}$, $\sigma \in \Sigma$ and $t \in \mathbb{R}^+$.*
$W^1 \xrightarrow{\sigma} W^2$ iff $\exists s^1 \in (H)^{-1}(W^1)$ and $s^2 \in (H)^{-1}(W^2) : s^1 \xrightarrow{\sigma} s^2$.

Proof. ($\Rightarrow$) Follows directly from Definition 5.28.
($\Leftarrow$) Suppose that $s^1 \in (H)^{-1}(W^1), s^2 \in (H)^{-1}(W^2)$, and that $s^1 \xrightarrow{\sigma} s^2$. Let $s^3 \in (H)^{-1}(W^1)$, we must prove that $\exists s^4 \in (H)^{-1}(W^2) : s^3 \xrightarrow{\sigma} s^4$. As $s^3 \in (H)^{-1}(W^1)$ and $s^1 \in (H)^{-1}(W^1)$, by Proposition 5.27, $s^3 \equiv s^1$. As $s^1 \xrightarrow{\sigma} s^2$, Proposition 5.23 ensures that $\exists s^4 \in (H)^{-1}(W^2) : s^3 \xrightarrow{\sigma} s^4$. □

Proposition 5.32: *For each word $W \in \mathcal{H}$, $Post(W)$ is finite and effectively computable.*

Proof. Let $W \in \mathcal{H}$. The set of all W'' such that $W \longrightarrow_T W''$ is a finite set of words with the same number of 4-tuples than W. We form this set accumulating the words computed recursively as follows, using at each time the last W_{next} obtained (and starting from W):

- if the first letter of W_{next} contains 4-tuples whose second component is $\{0\}$ or $\{1\}$ or ... or $\{c_{\max}\}$, the following W_{next} is the word created as follows:
  1. the 4-tuples of this first letter whose second component is $\{c_{\max}\}$ are replaced by the same 4-tuples in which $\{c_{\max}\}$ is replaced by $]c_{\max}, +\infty[$,
  2. the other 4-tuples of this first letter are deleted from it (if it then becomes empty, it is omitted). A new set of 4-tuples is created as a new second letter: it will contain these same 4-tuples in which the second component is replaced by the immediately following region

$(]0,1[$ instead of $\{0\}$, $]1,2[$ instead of $\{1\}$, ..., $]c_{\max}-1, c_{\max}[$ instead of $\{c_{\max}-1\}$). The end of the word does not change.

- else, the following W_{next} is the word created as follows. The last letter of W_{next} is deleted. Its 4-tuples are modified to create a new set that will contain these same 4-tuples in which the second component is replaced by the immediately following region ($\{1\}$ instead of $]0,1[$, $\{2\}$ instead of $]1,2[, \ldots, \{c_{\max}\}$ instead of $]c_{\max}-1, c_{\max}[$). This new set is either joined with the first letter of the modified W_{next}, if it contains 4-tuples having $]c_{\max}, +\infty[$ as second components, or added as a new first letter of the modified W_{next} otherwise. The rest of the word does not change.

We stop when we encounter a W_{next} that has a unique letter whose 4-tuples have $]c_{\max}, +\infty[$ as second components.
Then, for each possible W'' such that $W \longrightarrow_T W''$, we easily find a $s \in S$ such that $H(s) = W''$ (note that the choice of s does not matter thanks to Proposition 5.30). For all $\sigma \in \Sigma$, it is easy to compute the set of elements s' of S such that $s \xrightarrow{\sigma} s'$. This set is finite and, from each of its elements s', we can get back $H(s')$.
Once we have examined each letter $\sigma \in \Sigma$, for each possible W'', the (finite !) set of all the $H(s')$ found form $Post(W)$. $\square$

APPENDIX E

Proof of Proposition 5.50

In this section, we present the proof of Proposition 5.50. It is closed to the proof of Proposition 4.70.

Proposition 5.50: *Let $\mathcal{Z}_m$ be a zone.*

$$[\![Post_D(\mathcal{Z}_m)]\!] = \{s' \mid \exists s \in \mathcal{Z}_m \text{ such that } s \to s' \text{ in } \mathcal{S}_{\mathcal{B},\neg\Phi}\}.$$

Proof. ($\subseteq$) Suppose that $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$, where $loc_{\mathcal{B}} = (b, mark_{\mathcal{B}})$ and $\forall 1 \leqslant k \leqslant m, loc_{\mathcal{A}}(x_k) = (\ell_k, mark_k)$. Let $s' \in Post_D(\mathcal{Z}_m)$. There exists a certain $s' \in \mathcal{Z}'_{m'}$ for a certain $\mathcal{Z}'_{m'} \in Post_D(\mathcal{Z}_m)$ constructed thanks to $\sigma, t_{\mathcal{B}}, t_1, \ldots, t_m$, and (without loss of generality) thanks to $r^{\ell_1}, \ldots, r^{\ell_p}$, representing the respective reset in locations $\ell_1, \ldots, \ell_p$. We note $\mathcal{Z}'_{m'} = (loc'_{\mathcal{A}}, loc'_{\mathcal{B}}, Z')$ and $t_{\mathcal{B}} = (b, \sigma, g, r, b')$. Let us suppose that $s' = \{(\ell'_{k'}, I'_{k'}, mark'_{k'})_{k'=1}^{m'}\} \cup \{(b', v', mark'_{\mathcal{B}})\}$, with, $\forall 1 \leqslant k' \leqslant m'$, $I'_{k'} = [v'(x_{k'}), v'(y_{k'})]$. We will construct a particular state s of $\mathcal{Z}_m$ and then prove that $s \xrightarrow{\sigma} s'$ in $\mathcal{S}_{\mathcal{B},\neg\Phi}$. Let us construct

$s = \{(\ell_k, I_k, mark_k)_{k=1}^m\} \cup \{(b, v, mark_{\mathcal{B}})\}$, with, $\forall 1 \leqslant k \leqslant m$, $I_k = [v(x_k), v(y_k)]$, where:

1. Arc of $\mathcal{B}$ without reset: $\forall 1 \leqslant i \leqslant n$: if $x_i^{\mathcal{B}} \notin r$, we define $v(x_i^{\mathcal{B}}) = v'(x_i^{\mathcal{B}})$,
2. New complete interval: $\forall 1 \leqslant k \leqslant m$: if there exists $1 \leqslant j \leqslant p$ such that r^{ℓ_j} is a doubloon and $x_k \in r^{\ell_j}$, then x^{ℓ_k} and y^{ℓ_k} were not used in $\mathcal{Z}$ and their values must not be defined,
3. Loop without merge: $\forall 1 \leqslant k \leqslant m$: if $\forall 1 \leqslant j \leqslant p$, $x_k \notin r^{\ell_j}$ but that $loc'_{\mathcal{A}}$ is defined on x_k, we define $v(x_k) = v'(x_k)$ and $v(y_k) = v'(y_k)$,
4. Loop with merge: $\forall 1 \leqslant k \leqslant m'$: if there exists $1 \leqslant j \leqslant p$ such that r^{ℓ_j} is a singleton and $x_k \in r^{\ell_j}$, then we define $v(y_k) = v'(y_k)$,
5. Arc going out or arc of $\mathcal{B}$ with reset: the values of $v(x_k)$, $v(y_k)$ and $v(x_i^{\mathcal{B}})$ that we still must define are arbitrarily chosen in way they satisfy the extended clock constraints of $g_{t_{\mathcal{B}},t_1,\ldots,t_m} \cap Z$ (which is possible because $g_{t_{\mathcal{B}},t_1,\ldots,t_m} \cap Z$ is satisfiable and we only chose values of clocks/clock copies that have the same value in $\mathcal{Z}'_{m'}$, so that they cannot prevent $g_{t_{\mathcal{B}},t_1,\ldots,t_m} \cap Z$ from being satisfiable).

We must prove that $s \xrightarrow{\sigma} s'$ in $\mathcal{S}_{\mathcal{B},\neg\Phi}$ (see Definition 5.18). Let us start proving case (a) of this definition, i.e.: $(b, v) \xrightarrow{\sigma} (b', v')$ in $\mathcal{B}$ and $\{(\ell_k, I_k)_{k=1}^m\} \xrightarrow{\sigma}_{f^\star_{\neg\Phi}} \{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ in $\mathcal{A}_{\neg\Phi}$.
We first show that $(b, v) \xrightarrow{\sigma} (b', v')$ in $\mathcal{B}$ thanks to $t_{\mathcal{B}}$. Indeed, $v \models g$ because g is contained in $g_{t_{\mathcal{B}},t_1,\ldots,t_m} \cap Z$; $\forall x \in r$, $v'(x) = 0$ because it is reset by definition of $\mathcal{Z}'_{m'} \in Post_D(\mathcal{Z}_m)$ and $\forall x \in \{x_1^{\mathcal{B}}, \ldots, x_2^{\mathcal{B}}\} \backslash r$, $v'(x) = v(x)$ by previous point 1..
Now, let us show that $\{(\ell_k, I_k)_{k=1}^m\} \xrightarrow{\sigma}_{f^\star_{\neg\Phi}} \{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ in $\mathcal{A}_{\neg\Phi}$. We must prove there exists minimal models M_k, for $1 \leqslant k \leqslant m$ of $\delta(\ell_k, \sigma)$ with respect to I_k such that $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\} \in f^\star_{\neg\Phi}\left((\bigcup_{k=1}^m M_k)^{\neq}\right)$. For each $1 \leqslant k \leqslant m$, we take M_k to be the minimal model of $\delta(\ell_k, \sigma)$ with respect to I_k obtained following the arc t_k: it can be taken because its clock constraint is verified on $v(x_k)$ and

$v(y_k)$ (it is indeed present in $g_{t_{\mathcal{B}},t_1,\ldots,t_m} \cap Z$) and as this clock constraint can only be an interval (convex), it is also verified on each $i \in I_k$. We then take the element E of $f^{\star}_{\neg\Phi}\left((\bigcup_{k=1}^{m} M_k)^{\neq}\right)$ that merges the two more little intervals present in ℓ_j, for $1 \leqslant j \leqslant p$, iff r^{ℓ_j} is a singleton. It remains to prove that the obtained configuration is exactly $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$. It is based on the following facts:

a. each (ℓ_k, I_k) that does not loop disappear from $\{(\ell_k, I_k)_{k=1}^{m}\}$ to E and is not present in $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$: the clock copies representing such intervals disappear from $\mathcal{Z}_m$ to $\mathcal{Z}'_{m'}$ (i.e. $loc_{\mathcal{A}}$ is not defined on them anymore),

b. for each location ℓ_j, with $1 \leqslant j \leqslant p$, destination of at least one t_k for $1 \leqslant k \leqslant m$, $(\ell_j, [0,0])$ is present in $(\bigcup_{k=1}^{m} M_k)^{\neq}$ and

   - <u>if r^{ℓ_j} is a doubloon:</u> two new clock copies, say x_{ℓ_j} and y_{ℓ_j} are used by $\mathcal{Z}'_{m'}$. They are defined and reset in way $(\ell_j, [0,0])$ is also present in $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$;
   - <u>else:</u> by definition of r^{ℓ_j} and $minLoop(t_1, \ldots, t_m)$, the clock copy x_j representing the beginning of the more little interval in $\{(\ell_k, I_k)_{k=1}^{m}\}$ that loops on ℓ_j, say $I_j = [v(x_j), v(y_j)]$, is reset in $\mathcal{Z}'_{m'}$: $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ contains $(\ell_j, [0, v(y_j)])$;

c. each (ℓ_k, I_k) that loops is still present in $(\bigcup_{k=1}^{m} M_k)^{\neq}$: the clock copies representing I_k in $\mathcal{Z}_m$ are still present in $\mathcal{Z}'_{m'}$ but the clock copy representing its beginning could have been reset, so that $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ contains either $(\ell_k, [v(x_k), v(y_k)])$ or $(\ell_k, [0, v(y_k)])$,

d. when computing E from $(\bigcup_{k=1}^{m} M_k)^{\neq}$, we know the two more little intervals present in ℓ_j, for $1 \leqslant j \leqslant p$, are merged iff r^{ℓ_j} is a singleton. So, $(\ell_j, [0,0])$ and (ℓ_j, I_j) are merged in $(\ell_j, [0, v(y_j)])$ iff I_j is the more little interval that loops on ℓ_j, r^{ℓ_j} is a singleton and so must contain the clock copy representing its beginning: x_j. In this case and only in this case, x_j is then reset by definition of $\mathcal{Z}'_{m'} \in Post_D(\mathcal{Z}_m)$ so that E contains $(\ell_j, [0, v(y_j)])$ iff $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ also contains $(\ell_j, [0, v(y_j)])$.

It remains to prove cases (b), (c) and (d) of Definition 5.18, ruling the evolution of Miyano-Hayashi markers. The Miyano-Hayashi markers are treated in the same way from s to s' than from $\mathcal{Z}_m$ to $\mathcal{Z}'_{m'}$ (observing Definition 5.18 and the definition of the $\mathcal{Z}'_{m'} \in Post_D(\mathcal{Z}_m)$, when $\mathcal{Z}'_{m'}$ is induced by $\sigma, t_{\mathcal{B}}, t_1, \ldots, t_m, r^{\ell_1}, \ldots, r^{\ell_p}$). Remark that s and $\mathcal{Z}_m$ are always accepting in the same time. When they are accepting, all the markers of both s and $\mathcal{Z}_m$ are turned to $\perp$. Then, they are treated as non-accepting state/zone and their markers are kept updated as follows.

e. The Miyano-Hayashi marker $mark'_{\mathcal{B}}$ of s' is $\top$ iff $mark_{\mathcal{B}} = \top$ or $b' \in F^{\mathcal{B}}$ (see Definition 5.18); while the marker of $loc'_{\mathcal{B}}$ of $\mathcal{Z}'_{m'}$ is $\top$ iff the location of $loc'_{\mathcal{B}}$ (i.e. b') is in $F^{\mathcal{B}}$ or $(b', \top) \in LOCM(t_{\mathcal{B}}, t_1, \ldots, t_m)$.

f. In s', for all $1 \leqslant k' \leqslant m'$: $\left(\ell'_{k'} \in F \;\Rightarrow\; mark'_{k'} = \top\right)$ (see Definition 5.18); while the marker $mark'_{k'}$ of $\mathcal{Z}'_{m'}$ is always marked by $\top$ when the location of $loc'_{\mathcal{A}}(x'_{k'})$ is in F.

g. In s', $\forall 1 \leqslant \overline{k} \leqslant m'$ with $\ell'_{\overline{k}} \notin F$: $mark'_{\overline{k}} = \perp$ iff $\exists 1 \leqslant k^* \leqslant m$ such that $(\ell_{k^*}, I_{k^*}, \perp) \in s$ and $(\ell'_{\overline{k}}, I'_{\overline{k}}) \in \mathsf{dest}(\{(\ell_k, I_k)_{k \in K}\}, \{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}, (\ell_{k^*}, I_{k^*}))$ (see Definition 5.18). The same marker $mark'_{\overline{k}}$ of $\mathcal{Z}'_{m'}$ is $\perp$ iff

   (*i*) $x_{\overline{k}} \in Loop(t_1, \ldots, t_m) \setminus \left(\bigcup_{i=1}^{p} r^{\ell_i}\right)$, $loc'_{\mathcal{A}}(x_{\overline{k}})$ (i.e. $\ell'_{\overline{k}}$) is not in F and the marker of $loc_{\mathcal{A}}(x_{\overline{k}})$ is $\perp$, or

   (*ii*) $x_{\overline{k}} \in \left(r^{\ell'_{\overline{k}}} \cap Copies_{begin}\right) \setminus Loop(t_1, \ldots, t_m)$ and $(\ell'_{\overline{k}}, \perp) \in LOCM(t_{\mathcal{B}}, t_1, \ldots, t_m)$, or

   (*iii*) $x_{\overline{k}} \in r^{\ell'_{\overline{k}}} \cap Loop(t_1, \ldots, t_m)$, and $loc'_{\mathcal{A}}(x_{\overline{k}})$ (i.e. $\ell'_{\overline{k}}$) is not in F, and either $(\ell, \perp) \in LOCM(t_{\mathcal{B}}, t_1, \ldots, t_m)$ or the marker of $\ell'_{\overline{k}}$ is $\perp$.

   From s to s' as well as from $\mathcal{Z}_m$ to $\mathcal{Z}'_{m'}$, this last case reflects the fact that a marker $\perp$ is associated to an interval (or a pair of clock copies for $\mathcal{Z}_m$) if either (*i*) this interval loops without merging on a non accepting location and was already marked $\perp$ or (*ii*) this interval results from the creation of a new interval in a location and this creation is due to the changing of

location of at least one interval that was marked $\perp$ in its source location, or (iii) this interval results from the creation of a 'semi-new interval' (a new interval is merged with the previous smallest interval of this location) and either the previous smallest interval of this location was marked $\perp$, or the creation of this semi-new interval is due to the changing of location of at least one interval that was marked $\perp$ in its source location.

($\supseteq$) Let $\mathcal{Z}_m = (loc_{\mathcal{A}}, loc_{\mathcal{B}}, Z)$ be a zone and s' be such that $\exists s \in \mathcal{Z}_m$ with $s \to s'$ in $\mathcal{S}_{\mathcal{B},\neg\Phi}$. Let us show that $s' \in Post_D(\mathcal{Z}_m)$. Let us suppose that $s = \{(\ell_k, I_k, mark_k)_{k=1}^m\} \cup \{(b, v, mark_{\mathcal{B}})\}$, with, $\forall 1 \leqslant k \leqslant m$, $I_k = [v(x_k), v(y_k)]$ and $s' = \{(\ell'_{k'}, I'_{k'}, mark'_{k'})_{k'=1}^{m'}\} \cup \{(b', v', mark'_{\mathcal{B}})\}$, with, $\forall 1 \leqslant k' \leqslant m'$, $I'_{k'} = [v'(x_{k'}), v'(y_{k'})]$. As $s \to s'$, there exists $\sigma \in \Sigma$ such that:

- $(b, v) \xrightarrow{\sigma} (b', v')$ in $\mathcal{B}$, i.e.: there exists an arc $t_{\mathcal{B}} = (b, \sigma, g, r, b')$ such that $v \models g$, $\forall x \in r$, $v'(x) = 0$ and $\forall x \in \{x_1^{\mathcal{B}}, \ldots, x_n^{\mathcal{B}}\} \backslash r$, $v'(x) = v(x)$;
- $\{(\ell_k, I_k)_{k=1}^m\} \xrightarrow{\sigma}_{f^\star_{\neg\Phi}} \{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\}$ in $\mathcal{A}_{\neg\Phi}$, i.e.: $\{(\ell'_{k'}, I'_{k'})_{k'=1}^{m'}\} = E \in f^\star_{\neg\Phi}\left((\bigcup_{k=1}^m M_k)^{\neq}\right)$ for certain minimal models M_k of $\delta(\ell_k, \sigma)$ with respect to I_k, which are themselves obtained by taking certain arcs t_k from ℓ_k, for $1 \leqslant k \leqslant m$.

Let us define r^{ℓ_j}, for $1 \leqslant j \leqslant p$ by:

- r^{ℓ_j} contains two new clock copies *iff* $\exists 1 \leqslant k \leqslant m$ such that t_k goes to ℓ_j with a reset and no merge is applied by $f^\star_{\neg\Phi}$ on ℓ_j,
- r^{ℓ_j} contains the clock representing the beginning of the more little interval present in ℓ_j that loops on ℓ_j taking one of the t_k,for $1 \leqslant k \leqslant m$, *iff* $\exists 1 \leqslant k \leqslant m$ such that t_k goes to ℓ_j with a reset and a merge is applied by $f^\star_{\neg\Phi}$ on ℓ_j,
- r^{ℓ_j} is undefined in the other case, i.e. when none of the t_k, for $1 \leqslant k \leqslant m$, goes to ℓ_j with a reset.

It is easy to prove that the $\mathcal{Z}'_{m'} \in Post_D(\mathcal{Z}_m)$ induced by $\sigma, t_{\mathcal{B}}, t_1, \ldots, t_m, r^{\ell_1}, \ldots, r^{\ell_p}$ contains s', noting $\mathcal{Z}'_{m'} = (loc'_{\mathcal{A}}, loc'_{\mathcal{B}}, Z')$, thanks to the following facts.

- As $s \in \mathcal{Z}_m$, the extended constraint Z is satisfied by its clock values ; moreover, having taken arcs $t_{\mathcal{B}}, t_1, \ldots, t_m$ ensure the bounds of the intervals and the clock values of s satisfy $g_{t_{\mathcal{B}}, t_1, \ldots, t_m} \cap Z$ (in particular $g_{t_{\mathcal{B}}, t_1, \ldots, t_m} \cap Z$ is satisfiable and $\mathcal{Z}'_{m'}$ really exists).
- A clock $x_i^{\mathcal{B}}$, for $1 \leqslant i \leqslant n$, is reset in the construction of $\mathcal{Z}'_{m'}$ iff $v'(x_i^{\mathcal{B}}) = 0$.
- The facts b., c. and d. of the proof of inclusion $\subseteq$ are still true here.
- The Miyano-Hayashi markers are treated in the same way from $\mathcal{Z}_m$ to $\mathcal{Z}'_{m'}$ than from s to s' (observing Definition 5.18 and the definition of the $\mathcal{Z}'_{m'} \in Post_D(\mathcal{Z}_m)$, when $\mathcal{Z}'_{m'}$ is induced by $\sigma, t_{\mathcal{B}}, t_1, \ldots, t_m, r^{\ell_1}, \ldots, r^{\ell_p}$): see cases e., f. and g. of the proof of inclusion $\subseteq$ for details.

$\square$

Bibliography

[1] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings*, volume 372 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1989.
2 citations in pages 5 and 33.

[2] P. A. Abdulla, J. Deneux, J. Ouaknine, K. Quaas, and J. Worrell. Universality analysis for one-clock timed automata. *Fundam. Inform.*, 89(4):419–450, 2008.
2 citations in pages 186 and 250.

[3] R. Alur and D. L. Dill. Automata for modeling real-time systems. In M. Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.
Cited in page 36.

[4] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
7 citations in pages 3, 36, 41, 42, 43, 151, and 186.

[5] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.
8 citations in pages 4, 90, 92, 96, 97, 107, 127, and 234.

[6] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999.
Cited in page 101.

[7] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.
4 citations in pages 3, 61, 92, and 95.

[8] R. Alur and P. Madhusudan. Decision problems for timed automata: A survey. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2004.
Cited in page 186.

[9] E. Asarin and P. Bouyer, editors. *Formal Modeling and Analysis of Timed Systems, 4th International Conference, FORMATS 2006, Paris, France, September 25-27, 2006, Proceedings*, volume 4202 of *Lecture Notes in Computer Science*. Springer, 2006.
2 citations in pages 377 and 379.

[10] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
2 citations in pages 2 and 18.

[11] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, ad-*

ditional chapters have been commissioned], volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.
2 citations in pages 190 and 193.

[12] N. Bertrand, A. Stainer, T. Jéron, and M. Krichen. A game approach to determinize timed automata. In M. Hofmann, editor, *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2011.
Cited in page 42.

[13] D. Berwanger, K. Chatterjee, M. D. Wulf, L. Doyen, and T. A. Henzinger. Alpaga: A tool for solving parity games with imperfect information. In S. Kowalewski and A. Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*, pages 58–61. Springer, 2009.
Cited in page 3.

[14] A. Bohy. *Antichain based algorithms for the synthesis of reactive systems.* PhD thesis, Computer Science Department. University of Mons, Belgium, may, 2014.
Cited in page 332.

[15] P. Bouyer. Timed automata may cause some troubles. Technical report, Research Report LSV-02-9, Lab. Spécification et Vérification, CNRS & ENS de Cachan, France, 2002.
3 citations in pages 201, 202, and 261.

[16] P. Bouyer. Model-checking timed temporal logics. *Electr. Notes Theor.*

Comput. Sci., 231:323–341, 2009.
4 citations in pages 62, 63, 92, and 93.

[17] P. Bouyer, L. Bozzelli, and F. Chevalier. Controller synthesis for MTL specifications. In C. Baier and H. Hermanns, editors, *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings*, volume 4137 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2006.
18 citations in pages 5, 6, 76, 77, 80, 84, 85, 281, 282, 283, 288, 299, 303, 307, 308, 309, 317, and 320.

[18] T. Brihaye, V. Bruyère, L. Doyen, M. Ducobu, and J. Raskin. Antichain-based QBF solving. In T. Bultan and P. Hsiung, editors, *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*, volume 6996 of *Lecture Notes in Computer Science*, pages 183–197. Springer, 2011.
Cited in page 167.

[19] T. Brihaye, M. Estiévenart, and G. Geeraerts. On MITL and alternating timed automata. In V. A. Braberman and L. Fribourg, editors, *Formal Modeling and Analysis of Timed Systems - 11th International Conference, FORMATS 2013, Buenos Aires, Argentina, August 29-31, 2013. Proceedings*, volume 8053 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2013.
3 citations in pages 6, 108, and 128.

[20] T. Brihaye, M. Estiévenart, and G. Geeraerts. On MITL and alternating timed automata. *CoRR*, abs/1304.2814, 2013.
3 citations in pages 6, 108, and 128.

[21] T. Brihaye, M. Estiévenart, and G. Geeraerts. On MITL and alternating timed automata over infinite words. In A. Legay and M. Bozga, editors, *Formal Modeling and Analysis of Timed Systems - 12th International Conference, FORMATS 2014, Florence, Italy, September 8-10, 2014. Proceedings*,

volume 8711 of *Lecture Notes in Computer Science*, pages 69–84. Springer, 2014.
2 citations in pages 6 and 207.

[22] T. Brihaye, M. Estiévenart, and G. Geeraerts. On MITL and alternating timed automata over infinite words. *CoRR*, abs/1406.4395, 2014.
2 citations in pages 6 and 207.

[23] P. E. Bulychev, A. David, K. G. Larsen, and G. Li. Efficient controller synthesis for a fragment of $\mathrm{mtl}_{0,\infty}$. *Acta Inf.*, 51(3-4):165–192, 2014.
Cited in page 323.

[24] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In E. Brinksma and K. G. Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2002.
2 citations in pages 2 and 32.

[25] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.
Cited in page 31.

[26] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS'96), December 4-6, 1996, Washington, DC, USA*, pages 73–81. IEEE Computer Society, 1996.
Cited in page 263.

[27] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989,*

Proceedings, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.
3 citations in pages 186, 187, and 251.

[28] L. Doyen, G. Geeraerts, J. Raskin, and J. Reichert. Realizability of real-time logics. In J. Ouaknine and F. W. Vaandrager, editors, *Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings*, volume 5813 of *Lecture Notes in Computer Science*, pages 133–148. Springer, 2009.
4 citations in pages 77, 101, 102, and 282.

[29] L. Doyen and J. Raskin. Improved algorithms for the automata-based approach to model-checking. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4424 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2007.
Cited in page 3.

[30] L. Doyen and J. Raskin. Antichain algorithms for finite automata. In J. Esparza and R. Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 2–22. Springer, 2010.
Cited in page 167.

[31] D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In H. Alt and A. Ferreira, editors, *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings*, volume 2285 of *Lecture Notes*

in Computer Science, pages 571–582. Springer, 2002.
10 citations in pages 5, 76, 77, 80, 85, 283, 285, 288, 305, and 306.

[32] E. Filiot, N. Jin, and J. Raskin. Antichains and compositional algorithms for LTL synthesis. *Formal Methods in System Design*, 39(3):261–296, 2011.
3 citations in pages 5, 33, and 167.

[33] O. Finkel. Undecidable problems about timed automata. In Asarin and Bouyer [9], pages 187–199.
Cited in page 42.

[34] P. Gastin and D. Oddoux. Fast LTL to büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.
4 citations in pages 32, 33, 34, and 212.

[35] G. Geeraerts, G. Kalyon, T. L. Gall, N. Maquet, and J. Raskin. Lattice-valued binary decision diagrams. In A. Bouajjani and W. Chin, editors, *Automated Technology for Verification and Analysis - 8th International Symposium, ATVA 2010, Singapore, September 21-24, 2010. Proceedings*, volume 6252 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2010.
Cited in page 265.

[36] B. D. Giampaolo, G. Geeraerts, J. Raskin, and N. Sznajder. Safraless procedures for timed specifications. In K. Chatterjee and T. A. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, volume 6246 of *Lecture Notes in Computer Science*, pages 2–22. Springer, 2010.
Cited in page 5.

[37] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research.* Springer-Verlag New York, Inc., New

York, NY, USA, 2002.
4 citations in pages 2, 14, 16, and 20.

[38] T. A. Henzinger, J. Raskin, and P. Schobbens. The regular real-time languages. In K. G. Larsen, S. Skyum, and G. Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 1998.
Cited in page 97.

[39] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
2 citations in pages 3 and 57.

[40] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001.
2 citations in pages 211 and 212.

[41] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997.
3 citations in pages 3, 36, and 186.

[42] S. Lasota and I. Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Log.*, 9(2), 2008.
3 citations in pages 43, 44, and 108.

[43] C. Löding and W. Thomas. Alternating automata and logics over infinite words. In J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, and T. Ito, editors, *Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, International Conference IFIP TCS 2000, Sendai, Japan, August 17-19, 2000, Proceedings*, volume 1872 of *Lecture Notes in Computer Science*, pages 521–535. Springer, 2000.
Cited in page 212.

[44] O. Maler, D. Nickovic, and A. Pnueli. From MITL to timed automata. In

Asarin and Bouyer [9], pages 274–289.
2 citations in pages 4 and 90.

[45] N. Maquet. *New Algorithms and Data Structures for the Emptiness Problem of Alternating Automata.* PhD thesis, Université Libre de Bruxelles, 2011.
3 citations in pages 34, 107, and 248.

[46] S. Miyano and T. Hayashi. Alternating finite automata on omega-words. *Theor. Comput. Sci.*, 32:321–330, 1984.
2 citations in pages 26 and 208.

[47] D. E. Muller, A. Saoudi, and P. E. Schupp. Alternating automata, the weak monadic theory of trees and its complexity. *Theor. Comput. Sci.*, 97(2):233–244, 1992.
2 citations in pages 211 and 212.

[48] J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 54–63. IEEE Computer Society, 2004.
Cited in page 43.

[49] J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 188–197. IEEE Computer Society, 2005.
2 citations in pages 44 and 108.

[50] J. Ouaknine and J. Worrell. On metric temporal logic and faulty turing machines. In L. Aceto and A. Ingólfsdóttir, editors, *Foundations of Software Science and Computation Structures, 9th International Conference, FOSSACS 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25-31, 2006, Proceedings*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–

230. Springer, 2006.
5 citations in pages 61, 62, 65, 95, and 96.

[51] J. Ouaknine and J. Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3(1), 2007.
24 citations in pages 4, 5, 34, 44, 45, 55, 61, 62, 65, 85, 86, 87, 95, 134, 151, 152, 207, 208, 209, 213, 220, 235, 335, and 337.

[52] P. Parys and I. Walukiewicz. Weak alternating timed automata. *Logical Methods in Computer Science*, 8(3), 2012.
4 citations in pages 55, 56, 212, and 213.

[53] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.
Cited in page 2.

[54] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989.
3 citations in pages 5, 33, and 34.

[55] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, april 1959. ISSN 0018-8646. doi: 10.1147/rd.32.0114.
Cited in page 16.

[56] J. Raskin. *Logics, Automata and Classical Theories for Deciding Real Time*. PhD thesis, Computer Science Department. University of Namur, Belgium, april, 1999.
3 citations in pages 97, 98, and 101.

[57] J. Raskin and P. Schobbens. The logic of event clocks - decidability, complexity and expressiveness. *Journal of Automata, Languages and Combinatorics*,

4(3):247–286, 1999.
3 citations in pages 98, 100, and 101.

[58] T. C. Ruys and G. J. Holzmann. Advanced SPIN tutorial. In S. Graf and L. Mounier, editors, *Model Checking Software, 11th International SPIN Workshop, Barcelona, Spain, April 1-3, 2004, Proceedings*, volume 2989 of *Lecture Notes in Computer Science*, pages 304–305. Springer, 2004.
2 citations in pages 2 and 32.

[59] S. Safra. Exponential determinization for omega-automata with a strong fairness acceptance condition. *SIAM J. Comput.*, 36(3):803–814, 2006.
Cited in page 18.

[60] P. Schobbens, J. Raskin, and T. A. Henzinger. Axioms for real-time logics. *Theor. Comput. Sci.*, 274(1-2):151–182, 2002.
Cited in page 97.

[61] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.
2 citations in pages 31 and 32.

[62] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. M. Birtwistle, editors, *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, August 27 - September 3, 1995, Proceedings)*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1995.
Cited in page 34.

[63] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings of the Symposium on Logic in Computer Science (LICS'86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 332–344. IEEE Computer Society, 1986.
Cited in page 31.

[64] M. D. Wulf, L. Doyen, T. A. Henzinger, and J. Raskin. Antichains: A new algorithm for checking universality of finite automata. In T. Ball and R. B.

Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2006.
2 citations in pages 3 and 167.

[65] M. D. Wulf, L. Doyen, N. Maquet, and J. Raskin. Alaska. In S. D. Cha, J. Choi, M. Kim, I. Lee, and M. Viswanathan, editors, *Automated Technology for Verification and Analysis, 6th International Symposium, ATVA 2008, Seoul, Korea, October 20-23, 2008. Proceedings*, volume 5311 of *Lecture Notes in Computer Science*, pages 240–245. Springer, 2008.
Cited in page 3.

[66] M. D. Wulf, L. Doyen, N. Maquet, and J. Raskin. Antichains: Alternative algorithms for LTL satisfiability and model-checking. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2008.
2 citations in pages 3 and 167.