INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Game Approach to Determinize Timed Automata

Nathalie Bertrand — Amélie Stainer — Thierry Jéron — Moez Krichen

N° 7381

September 2010

Embedded and Real Time Systems

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7381--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche RENNES - BRETAGNE ATLANTIQUE

A Game Approach to Determinize Timed Automata

Nathalie Bertrand, Amélie Stainer, Thierry Jéron, Moez Krichen

Theme : Embedded and Real Time Systems
Algorithmics, Programming, Software and Architecture
Équipe-Projet Vertecs

Rapport de recherche n° 7381 — September 2010 — 22 pages

Abstract: Timed automata are frequently used to model real-time systems. Their determinization is a key issue for several validation problems. However, not all timed automata can be determinized, and determinizability itself is undecidable. In this paper, we propose a game-based algorithm which, given a timed automaton, tries to produce a language-equivalent deterministic timed automaton, otherwise a deterministic over-approximation. Our method subsumes two recent contributions: it is at once more general than the determinization procedure of [BBBB09a] and more precise than the approximation algorithm of [KT09]. Moreover, we explain how to extend our method to deal with invariants and ε-transitions, and also consider other useful approximations: under-approximation, and combination of under- and over-approximations.

Key-words: timed automata, determinization

This work was partially funded by the french ANR Testec

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Un jeu pour déterminiser des automates temporisés

Résumé : Les automates temporisés sont classiquement utilisés pour modéliser des systèmes temps-réel. Leur déterminisation est un problème clef pour plusieurs problèmes de validation. Pourtant certains automates temporisés ne sont pas déterminisables et la déterminisabilité elle-même est indécidable. Dans cet article, nous proposons un algorithme qui, étant donné un automate temporisé, essaie de produire un automate déterministe équivalent, ou à défaut une surapproximation déterministe. Notre méthode améliore deux contributions récentes dans le domaine : elle est à la fois plus générale que la procédure de déterminisation de [BBBB09a] et plus précise que l'algorithme d'approximation de [KT09]. De plus, nous expliquons comment étendre notre méthode pour traiter les invariants et les ε-transitions, et considérons également d'autres types d'approximation utiles : les sous-approximations et les combinaisons de sous- et sur-approximations.

Mots-clés : automates temporisés, déterminisation

1 Introduction

Timed automata (TA), introduced in [AD94], form a usual model for the specification of real-time embedded systems. Essentially TAs are an extension of automata with guards and resets of continuous clocks. They are extensively used in the context of many validation problems such as verification, control synthesis or model-based testing. One of the reasons for this popularity is that, despite the fact that they represent infinite state systems, their reachability is decidable, thanks to the construction of the region graph abstraction.

Determinization is a key issue for several problems such as implementability, diagnosis or test generation, where the underlying analyses depend on the observable behavior. In the context of timed automata, determinization is problematic for two reasons. First, determinizable timed automata form a strict subclass of timed automata [AD94]. Second, the problem of the determinizability of a timed automaton, (i.e. does there exist a deterministic TA with the same language as a given non-deterministic one?) is undecidable [Fin06, Tri06].

Therefore, in order to determinize timed automata, two alternatives have been investigated: either restricting to determinizable classes or choosing to ensure termination for all TAs by allowing over-approximations, i.e. deterministic TAs accepting more timed words. For the first approach, several classes of determinizable TAs have been identified, such as strongly non-Zeno TAs [AMPS98], event-clock TAs [AFH94], or TAs with integer resets [SPKM08]. In a recent paper, Baier, Bertrand, Bouyer and Brihaye [BBBB09a] propose a procedure which does not terminate in general, but allows one to determinize TAs in a class covering all the aforementioned determinizable classes. It is based on an unfolding of the TA into a tree, which introduces a new clock at each step, representing original clocks by a mapping; a symbolic determinization using the region abstraction; a folding up by the removal of redundant clocks. To our knowledge, the second approach has only been investigated by Krichen and Tripakis [KT09]. They propose an algorithm that produces a deterministic over-approximation based on a simulation of the TA by a deterministic TA with fixed resources (number of clocks and maximal constant). Its locations code (over-approximate) estimates of possible states of the original TA, and it uses a fixed policy governed by a finite automaton for resetting clocks.

We propose a method that combines techniques from and improves the approaches of [BBBB09a] and [KT09], despite their notable differences. It is inspired by a game-based approach to decide the diagnosability of TAs with fixed resources presented by Bouyer, Chevalier and D'Souza in [BCD05]. Similarly to [KT09], in our approach, the resulting deterministic TA is given fixed resources (number of clocks and maximal constant) in order to simulate the original TA by a coding of relations between new clocks and original ones. The core principle is the construction of a finite turn-based safety game between two players, Spoiler and Determinizator, where Spoiler chooses an action and the region of its occurrence, while Determinizator chooses which clocks to reset. Our main result states that if Determinizator has a winning strategy, then it yields a deterministic timed automaton accepting exactly the same timed language as the initial automaton, otherwise it produces a deterministic over-approximation.

Our approach is more general than the procedure of [BBBB09a], thus allowing one to enlarge the set of timed automata that can be automatically determinized, thanks to an increased expressive power in the coding of relations between new and original clocks, and robustness to some language inclusions. Moreover, in contrast to [BBBB09a], our techniques apply to a large class of timed automata: TA with ε-transitions and invariants. It is also more precise than the algorithm of [KT09] in several respects: an adaptative and timed resetting policy, governed by a strategy, compared to a fixed untimed one and a more precise update of the relations between clocks, even for a fixed policy, allow our method to be exact on a larger class of TAs. The model used in [KT09] includes silent transitions, and edges are labeled with urgency status (eager, delayable, or lazy), but urgency is not preserved by their over-approximation algorithm. These observations illustrate the benefits of our game-based approach compared to existing work.

Another contribution is the generalization of our game-based approach to generate deterministic under-approximations or deterministic approximations combining under- and over-approximations. The motivation for this generalization is to tackle the problem of off-line model-based test generation from non-deterministic timed automata specifications. Indeed in this context, input actions and output actions have to be considered differently to build the approximation. We provide a notion of refinement (and the dual abstraction) and explain how to extend our approach to generate deterministic abstractions.

The structure of this report is as follows. In Section 2 we recall definitions and properties relative to timed automata, and present the two recent pieces of work to determinize timed automata or provide a deterministic over-approximation. Section 3 is devoted to the presentation of our game approach and its properties. A comparison with existing methods is detailed in Section 4. Extensions of the method to timed automata with invariants and ε-transitions are then presented in Section 5. In Section 6, we finally discuss how the construction can be adapted to perform under-approximations, or combinations of under- and over-approximations.

2 Preliminaries

In this section, we start by introducing the model of timed automata, and then review two approaches for their determinization.

2.1 Timed Automata

We start by introducing notations and useful definitions concerning timed automata [AD94].

Given a finite set of clocks X, a clock *valuation* is a mapping $v : X \to \mathbb{R}_+$. We note $\overline{0}$ the valuation that assigns 0 to all clocks. If v is a valuation over X and $t \in \mathbb{R}_+$, then $v+t$ denotes the valuation which assigns to every clock $x \in X$ the value $v(x)+t$, and $\overrightarrow{v} = \{v+t \mid t \in \mathbb{R}\}$ denotes past and future timed extensions of v. For $X' \subseteq X$ we write $v_{[X' \leftarrow 0]}$ for the valuation equal to v on $X \setminus X'$ and to $\overline{0}$ on X', $v_{|X'}$ for the valuation v restricted to X', and $v_{[X' \leftarrow 0]^{-1}}$ for the set of valuations v' such that $v'_{[X \leftarrow 0]} = v$.

Given a non-negative integer M, an *M-bounded guard*, or simply guard when M is clear from context, over X is a finite conjunction of constraints of the form $x \sim c$ where $x \in X$, $c \in [0, M] \cap \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. We denote by $G_M(X)$ the set of M-bounded guards over X. Given a guard g and a valuation v, we write $v \models g$ if v satisfies g. Invariants are restricted cases of guards: given $M \in \mathbb{N}$, an *M-bounded invariant* over X is a finite conjunction of constraints of the form $x \lhd c$ where $x \in X$, $c \in [0, M] \cap \mathbb{N}$ and $\lhd \in \{<, \leq\}$. We denote by $I_M(X)$ the set of invariants. Given two finite sets of clocks X and Y, a *relation* between clocks of X and those of Y is a finite conjunction C of atomic constraints of the form $x - y \sim c$ where $x \in X$, $y \in Y$, $\sim \in \{<, =, >\}$ and $c \in \mathbb{N}$. When, moreover, the constant c is constrained to belong to $[-M', M]$, for some constants $M, M' \in \mathbb{N}$, we denote by $\mathsf{Rel}_{M,M'}(X, Y)$ the set of relations between X and Y.

Definition 1. *A timed automaton (TA for short) is a tuple $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E, \mathsf{Inv})$ such that: L is a finite set of locations, $\ell_0 \in L$ is the initial location, $F \subseteq L$ is the set of final locations, Σ is a finite alphabet, X is a finite set of clocks, $M \in \mathbb{N}$, $E \subseteq L \times G_M(X) \times (\Sigma \cup \{\varepsilon\}) \times 2^X \times L$ is a finite set of edges, and $\mathsf{Inv} : L \to I_M(X)$ is the invariant function.*

The constant M is called the maximal constant of $\mathcal{A}$, and we will refer to $(|X|, M)$ as the *resources* of $\mathcal{A}$. The semantics of a timed automaton $\mathcal{A}$ is given as a timed transition system $\mathcal{T}_\mathcal{A} = (S, s_0, S_F, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \to)$ where $S = L \times \mathbb{R}_+^X$ is the set of states, $s_0 = (\ell_0, \overline{0})$ is the initial state, $S_F = F \times \mathbb{R}_+^X$ is the set of final states, and $\to \subseteq S \times (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})) \times S$ is the transition relation composed of moves of the form $(\ell, v) \xrightarrow{\tau, a} (\ell', v')$ whenever there exists an edge $(\ell, g, a, X', \ell') \in E$ such that $v + \tau \models g \wedge \mathsf{Inv}(\ell)$, $v' = (v + \tau)_{[X' \leftarrow 0]}$ and $v' \models \mathsf{Inv}(\ell')$.

A *run* ρ of $\mathcal{A}$ is a finite sequence of moves starting in s_0, *i.e.*, $\rho = s_0 \xrightarrow{\tau_1, a_1} s_1 \cdots \xrightarrow{\tau_k, a_k} s_k$. Run ρ is said accepting if it ends in $s_k \in S_F$. A *timed word* over Σ is an element $(t_i, a_i)_{i \leq n}$ of $(\mathbb{R}_+ \times \Sigma)^*$ such that $(t_i)_{i \leq n}$ is non-decreasing. The timed word associated with ρ is $w = (t_{i_1}, a_{i_1}) \dots (t_{i_m}, a_{i_m})$ where $(a_i \in \Sigma$ iff $\exists n,\ a_i = a_{i_n})$ and $t_i = \sum_{j=1}^{i} \tau_j$. We write $\mathcal{L}(\mathcal{A})$ for the language of $\mathcal{A}$, that is the set of timed words w such that there exists an accepting run which reads w. We say that two timed automata $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* whenever $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.

A *deterministic* timed automaton (abbreviated DTA) $\mathcal{A}$ is a TA such that for every timed word w, there is at most one run in $\mathcal{A}$ reading w. $\mathcal{A}$ is *determinizable* if there exists a deterministic timed automaton $\mathcal{B}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. It is well-known that some timed automata are not determinizable [AD94]; moreover, the determinizability of timed automata is an undecidable problem, even with fixed resources [Tri06, Fin06].

Example An example of a timed automaton is depicted in Figure 1. This non-deterministic timed automaton has ℓ_0 as initial location (denoted by a pending incoming arrow), ℓ_3 as final location (denoted by a pending outgoing arrow) and accepts the language: $\mathcal{L}(\mathcal{A}) = \{(t_1, a)\cdots(t_n, a)(t_{n+1}, b) \,|\, t_{n+1} < 1\}$.

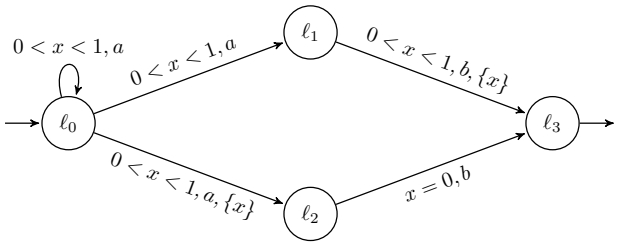

Figure 1: A timed automaton $\mathcal{A}$.

The region abstraction forms a partition of valuations over a given set of clocks. It allows one to make abstractions in order to decide properties like the reachability of a location. We let X be a finite set of clocks, and $M \in \mathbb{N}$. We write $\lfloor t \rfloor$ and $\{t\}$ for the integer part and the fractional part of a real t, respectively. The equivalence relation $\equiv_{X,M}$ over valuations over X is defined as follows: $v \equiv_{X,M} v'$ if (i) for every clock $x \in X$, $v(x) \leq M$ iff $v'(x) \leq M$; (ii) for every clock $x \in X$, if $v(x) \leq M$, then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ and $\{v(x)\} = 0$ iff $\{v'(x)\} = 0$ and (iii) for every pair of clocks $(x, y) \in X^2$ such that $v(x) \leq M$ and $v(y) \leq M$, $\{v(x)\} \leq \{v(y)\}$ iff $\{v'(x)\} \leq \{v'(y)\}$. The equivalence relation is called the *region equivalence* for the set of clocks X w.r.t. M, and an equivalence class is called a *region*. The set of regions, given X and M, is denoted Reg_M^X. A region r' is a *time-successor* of a region r if there is $v \in r$ and $t \in \mathbb{R}_+$ such that $v + t \in r'$. The set of all time-successors of r is denoted $\overrightarrow{r}$.

In the following, we often abuse notations for guards, invariants, relations and regions, and write g, I, C and r, respectively, for both the constraints over clock variables and the sets of valuations they represent.

2.2 Existing approaches to the determinization of TAs

To overcome the non-feasibility of determinization of timed automata in general, two alternatives have been explored: either exhibiting subclasses of timed automata which are determinizable and provide determinization algorithms, or constructing deterministic over-approximations. We relate here, for each of these directions, a recent contribution.

Determinization procedure. An abstract determinization procedure which effectively constructs a deterministic timed automaton for several classes of determinizable timed automata is presented in [BBBB09a]. Given a timed automaton $\mathcal{A}$, this procedure first produces a language-equivalent infinite timed tree, by unfolding $\mathcal{A}$, introducing a fresh clock at each step. This allows one to preserve all timing constraints, using a mapping from clocks of $\mathcal{A}$ to the new clocks. Then, the infinite tree is split into regions, and symbolically determinized. Under a clock-boundedness assumption, the infinite tree with infinitely many clocks can be folded up into a timed automaton (with finitely many locations and clocks). The clock-boundedness assumption is satisfied for several classes of timed automata, such as event-clock TAs [AFH94], TAs with integer resets [SPKM08] and strongly non-Zeno TAs [AMPS98], which can thus be determinized by this procedure. The resulting deterministic timed automaton is doubly exponential in the size of $\mathcal{A}$.

Deterministic over-approximation By contrast, Krichen and Tripakis propose in [KT09] an algorithm applicable to any timed automaton $\mathcal{A}$, which produces a deterministic over-approximation, that is a deterministic TA $\mathcal{B}$ accepting at least all timed words in $\mathcal{L}(\mathcal{A})$. This TA $\mathcal{B}$ is built by simulation of $\mathcal{A}$ using only information carried by clocks of $\mathcal{B}$. A location of $\mathcal{B}$ is then a state estimate of $\mathcal{A}$ consisting of a (generally infinite) set of pairs (ℓ, v) where ℓ is a location of $\mathcal{A}$ and v a valuation over the union of clocks of $\mathcal{A}$ and $\mathcal{B}$. This method is based on the use of a fixed finite automaton (called skeleton) which governs the resetting policy for the clocks of $\mathcal{B}$. $\mathcal{B}$ is also doubly exponential in the size of $\mathcal{A}$.

3 A game approach

In [BCD05], given a plant —modeled by a timed automaton— and fixed resources, the authors build a game where some player has a winning strategy if and only if the plant can be diagnosed by a timed automaton using the given resources. Inspired by this construction, given a timed automaton $\mathcal{A}$ and fixed resources, we derive a game between two players Spoiler and Determinizator, such that if Determinizator has a winning strategy, then a deterministic timed automaton $\mathcal{B}$ with $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A})$ can be effectively generated. Moreover, any strategy for Determinizator (winning or not) yields a deterministic over-approximation for $\mathcal{A}$. We present here the method for timed automa without ε-transitions and which all invariants are true. The general case is presented, for clarity, as extension in Section 5.

3.1 Definition of the game

Let $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E)$ be a timed automaton. We aim at building a deterministic timed automaton $\mathcal{B}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ if possible, or $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. In order to do so, we fix resources (k, M') for $\mathcal{B}$ and build a finite 2-player turn-based safety game $\mathcal{G}_{\mathcal{A},(k,M')}$. Players Spoiler and Determinizator alternate moves, and the objective of player Determinizator is to avoid a set of bad states (to be defined later). Intuitively, in the safe states, for sure, no over-approximation has been performed.

For simplicity, we first detail the approach in the case where $\mathcal{A}$ has no ε-transitions and all invariants are true.

Let Y be a set of clocks of cardinality k. The initial state of the game is a state of Spoiler consisting of location ℓ_0 (initial location of $\mathcal{A}$) together with the simplest relation between X and Y: $\forall x \in X$, $\forall y \in Y, x - y = 0$, and a marking $\top$ (no over-approximation was done so far), together with the null region over Y. In each of its states, Spoiler challenges Determinizator by proposing an M'-bounded region r over Y, and an action $a \in \Sigma$. Determinizator answers by deciding the set of clocks $Y' \subseteq Y$ he wishes to reset. The next state of Spoiler contains a region over Y ($r' = r_{[Y' \leftarrow 0]}$), and a finite set of configurations: triples formed of a location of $\mathcal{A}$, a relation between clocks in X and clocks in Y, and a boolean marking ($\top$ or $\bot$). A state of Spoiler thus constitutes a states estimate of $\mathcal{A}$, and the role of the markings is to indicate whether over-approximations possibly happened. A state of Determinizator is a copy of the preceding states estimate together with the move of Spoiler. Bad states player Determinizator wants to avoid are on the one hand states of the game where all configurations are marked $\bot$ and, on the other hand, states where all final configurations (if any) are marked $\bot$.

Formally, given $\mathcal{A}$ and (k, M') we define $\mathcal{G}_{\mathcal{A},(k,M')} = (\mathsf{V}, \mathsf{v}_0, \mathsf{Act}, \delta, \mathsf{Bad})$ where:

- The set of vertices V is partitioned into V_S and V_D, respectively vertices of Spoiler and Determinizator. Vertices of V_S and V_D are labeled respectively in $2^{L \times \mathsf{Rel}_{M,M'}(X,Y) \times \{\top,\bot\}} \times \mathsf{Reg}_{M'}^{Y}$ and $2^{L \times \mathsf{Rel}_{M,M'}(X,Y) \times \{\top,\bot\}} \times (\mathsf{Reg}_{M'}^{Y} \times \Sigma)$;
- $\mathsf{v}_0 = (\{(\ell_0, X - Y = 0, \top)\}, \{\overline{0}\})$ is the initial vertex and belongs to player Spoiler[1];
- $\mathsf{Act} = (\mathsf{Reg}_{M'}^{Y} \times \Sigma) \cup 2^Y$ is the set of possible actions;
- $\delta \subseteq \mathsf{V}_S \times (\mathsf{Reg}_{M'}^{Y} \times \Sigma) \times \mathsf{V}_D \cup \mathsf{V}_D \times 2^Y \times \mathsf{V}_S$ is the set of edges;
- $\mathsf{Bad} = \{(\{(\ell_j, C_j, \bot)\}_j, r)\} \cup \{(\{(\ell_j, C_j, b_j)\}_j, r) \,|\, \{\ell_j\}_j \cap F \neq \emptyset \wedge \forall j,\, \ell_j \in F \Rightarrow b_j = \bot\}$ is the set of bad states.

We now detail the edge relation which defines the possible moves of the players. Given $\mathsf{v}_S = (\mathcal{E}, r) \in \mathsf{V}_S$ a state of Spoiler and (r', a) one of its moves, the successor state is defined, provided r' is a time-successor of r, as the state $\mathsf{v}_D = (\mathcal{E}, (r', a)) \in \mathsf{V}_D$ if $\exists (\ell, C, b) \in \mathcal{E}$ and $\exists \ell \xrightarrow{g,a,X'} \ell' \in E$ s.t. $[r' \cap C]_{|X} \cap g \neq \emptyset$.

Given $\mathsf{v}_D = (\{(\ell_j, C_j, b_j)\}_j, (r', a)) \in \mathsf{V}_D$ a state of Determinizator and $Y' \subseteq Y$ one of its moves, the successor state of v_D is the state $(\mathcal{E}, r'_{[Y' \leftarrow 0]}) \in \mathsf{V}_S$ where $\mathcal{E}$ is obtained as the set of all elementary successors of configurations in $\{(\ell_j, C_j, b_j)\}_j$ by (r', a) and by resetting Y'. Precisely, if (ℓ, C, b) is a configuration, its elementary successors set by (r', a) and Y' is:

$$\mathsf{Succ}_e[r', a, Y'](\ell, C, b) = \left\{ (\ell', C', b') \;\middle|\; \begin{array}{l} \exists \ell \xrightarrow{g,a,X'} \ell' \in E \text{ s.t. } [r' \cap C]_{|X} \cap g \neq \emptyset \\ C' = \mathsf{up}(r', C, g, X', Y') \\ b' = b \wedge ([r' \cap C]_{|X} \subseteq g) \end{array} \right\}$$

[1] $X - Y = 0$ is a shortcut to denote the relation $\forall x \in X$, $\forall y \in Y$, $x - y = 0$.

where $\mathsf{up}(r', C, g, X', Y')$ is the update of the relation between clocks in X and Y after the moves of the two players, that is after taking action a in r', resetting $X' \subseteq X$ and $Y' \subseteq Y$, and forcing the satisfaction of g. Formally, $\mathsf{up}(r', C, g, X', Y') = \overleftrightarrow{(r' \cap C \cap g)_{[X' \leftarrow 0][Y' \leftarrow 0]}}$. Boolean b' is set to $\bot$ if either $b = \bot$ or the induced guard $[r' \cap C]_{|X}$ over-approximates g. In the update, the intersection with g aims at stopping runs that for sure will correspond to timed words out of $\mathcal{L}(\mathcal{A})$; the boolean b anyway takes care of keeping track of the possible over-approximation. Region r', relation C and guard g can all be seen as zones (*i.e.* unions of regions) over clocks $X \cup Y$. It is standard that elementary operations on zones, such as intersections, resets, future and past, can be performed effectively. As a consequence, the update of a relation can also be computed effectively.

Given the labeling of states in the game $\mathcal{G}_{\mathcal{A},(k,M')}$, the size of the game is doubly exponential in the size of $\mathcal{A}$. We will see in Subsection 3.3 that the number of edges in $\mathcal{G}_{\mathcal{A},(k,M')}$ can be impressively decreased, since restricting to atomic resets (resets of at most one clock at a time) does not diminish the power of Determinizator.

Example As an example, the construction of the game is illustrated on the non-deterministic timed automaton $\mathcal{A}$ depicted in Figure 1, for which we construct the associated game $\mathcal{G}_{\mathcal{A},(1,1)}$ represented in Figure 2. Rectangular states belong to Spoiler and circles correspond to states of Determinizator. Note that, for the sake of simplicity, the labels of states of Determinizator are omitted in the picture. Gray states form the set Bad. Let us detail the computation of the successors of the top left state by the move $((0,1), b)$ of Spoiler and moves ($\emptyset$ or $\{y\}$) of Determinizator. To begin with, note that b cannot be fired from ℓ_0 in $\mathcal{A}$, therefore the first configuration has no elementary successor. We then consider the configuration which contains the location ℓ_1. The guard induced by $x - y = 0$ and $y \in (0,1)$ is simply $0 < x < 1$ and the guard of the corresponding transition between ℓ_1 and ℓ_3 in $\mathcal{A}$ is exactly $0 < x < 1$, moreover this transition resets x. As a consequence, the successors states contain a configuration marked $\top$ with location ℓ_3 and, respectively, relations $-1 < x - y < 0$ and $x - y = 0$ for the two different moves of Determinizator. Last, when considering the configuration with location ℓ_2, we obtain elementary successors marked $\bot$. Indeed, the guard induced by this move of Spoiler and the relation $-1 < x - y < 0$ is $-1 < x < 1$ whereas the corresponding guard in $\mathcal{A}$ is $x = 0$. To preserve all timed words accepted by $\mathcal{A}$, we represent these configurations, but they imply over-approximations. Thus the successor states contain a configuration marked $\bot$ with location ℓ_3 and the same respective relations as before, thanks to the intersection with the initial guard $x = 0$ in $\mathcal{A}$.

3.2 Properties of the strategies

Given $\mathcal{A}$ a timed automaton and resources (k, M'), the game $\mathcal{G}_{\mathcal{A},(k,M')}$ is a finite-state safety game. It is well known that, for this kind of games, winning strategies can be chosen positional and they can be computed in linear time in the size of the arena [GTW02]. In the following, we simply write strategies for positional strategies. We will see in Subsection 3.3 that positional strategies (winning or not) are indeed sufficient in our framework. A strategy for player Determinizator thus assigns to each state $\mathsf{v}_D \in \mathsf{V}_D$ a set $Y' \subseteq Y$ of clocks to be reset; the successor state is then $\mathsf{v}_S \in \mathsf{V}_S$ such that $(\mathsf{v}_D, Y', \mathsf{v}_S) \in \delta$.

With every strategy for Determinizator σ we associate the timed automaton $\mathsf{Aut}(\sigma)$ obtained by merging a transition of Spoiler with the transition chosen by Determinizator just after, and setting final locations as states of Spoiler containing at least one final location of $\mathcal{A}$. If a strategy σ_S for Spoiler is fixed too, we denote by $\mathsf{Aut}(\sigma, \sigma_S)$ the resulting sub-automaton[2]. The main result of the paper is stated in the following theorem and links strategies of Determinizator with deterministic over-approximations of the initial timed language.

Theorem 1. *Let $\mathcal{A}$ be a timed automaton, and $k, M' \in \mathbb{N}$. For every strategy σ of Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$, $\mathsf{Aut}(\sigma)$ is a deterministic timed automaton over resources (k, M') and satisfies $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$. Moreover, if σ is winning, then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathsf{Aut}(\sigma))$.*

The full proof is given in the general case (with ε-transitions and invariants in $\mathcal{A}$) in Section 5.3; we however give below its main ideas.

[2] In the case where σ and/or σ_S have arbitrary memory, we abuse notation and write $\mathsf{Aut}(\sigma)$ and $\mathsf{Aut}(\sigma, \sigma_S)$ for the resulting potentially infinite objects.

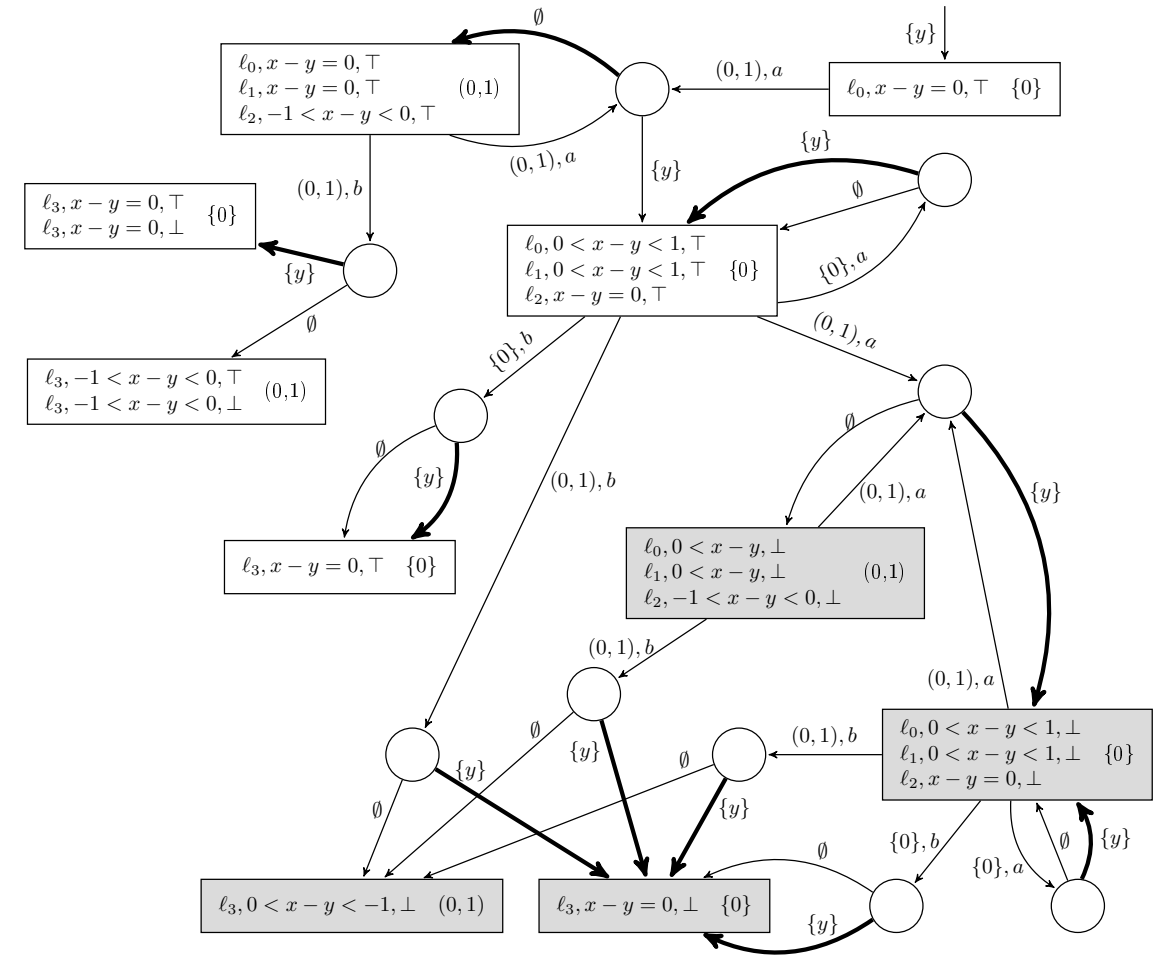

Figure 2: The game $\mathcal{G}_{\mathcal{A},(1,1)}$ and an example of winning strategy σ for Determinizator.

Sketch of proof. Given a strategy σ, we show that there exists a strong timed simulation between $\mathcal{A}$ and $\mathsf{Aut}(\sigma)$, namely the relation ρ defined as: $\rho = \{((\ell, v), ((\mathcal{E}, r), v')) \mid \exists (\ell, C, b) \in \mathcal{E},\ (v, v') \in C\}$. Hence $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$. Moreover, if σ is winning, ρ also satisfies the following condition which implies the other language inclusion: for all s', for all $a \in \Sigma$, whenever $s' \xrightarrow{\tau,a}_{\mathsf{Aut}(\sigma)} \tilde{s}'$ there exists s such that $(s, s') \in \rho$ and there exists $s \xrightarrow{\tau,a}_{\mathcal{A}} \tilde{s}$ with $(\tilde{s}, \tilde{s}') \in \rho$. □

Back to our running example, on Figure 2, a winning strategy for Determinizator is represented by the bold arrows. This strategy yields the deterministic equivalent for $\mathcal{A}$ depicted in Figure 3.

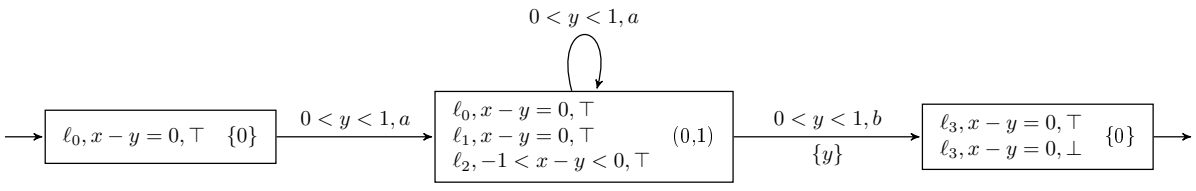

Figure 3: The deterministic TA $\mathsf{Aut}(\sigma)$ obtained by our construction.

Remark 1. *Because of the undecidability of the determinizability with fixed resources [Tri06, Fin06], contrary to the diagnosability problem, there is no hope to have a reciprocal statement to the one of*

Theorem 1 in the following sense: if $\mathcal{A}$ can be determinized with resources (k, M') then Determinizator has a winning strategy in $\mathcal{G}_{\mathcal{A},(k,M')}$. Figure 4 illustrates this phenomenon by presenting a timed automaton $\mathcal{A}$ which is determinizable with resources $(1,1)$, but for which all strategies for Determinizator in $\mathcal{G}_{\mathcal{A},(1,1)}$ are losing. Intuitively the self loop on ℓ_0 forces Determinizator to reset the clock in his first move; afterwards on each branch of the automaton (passing through ℓ_1, ℓ_2 or ℓ_3) the behavior of $\mathcal{A}$ is strictly over-approximated in the game. However, since these over-approximations cover each others, this losing strategy yields a deterministic equivalent to $\mathcal{A}$.

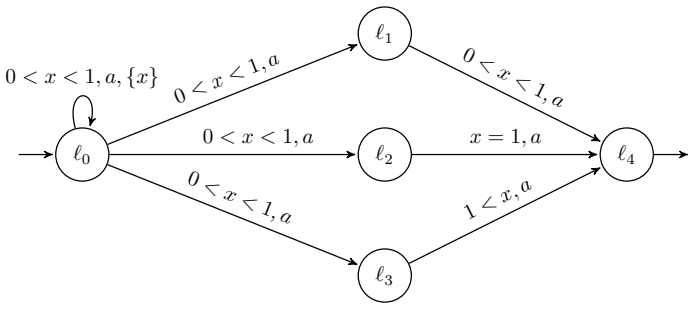

Figure 4: A determinizable TA for which there is no winning strategy for Determinizator.

3.3 Choosing a good losing strategy

Standard techniques allow one to check whether there is a winning strategy for Determinizator, and in the positive case, extract such a strategy [GTW02]. However, if Determinizator has no winning strategy to avoid the set of bad states, it is of interest to be able to choose a good losing strategy. To this aim, we introduce a natural partial order over the set of strategies of Determinizator based on the distance to the set Bad: $d_{\mathsf{Bad}}(\mathcal{A})$ denotes the minimal number of steps in some automaton $\mathcal{A}$ to reach Bad from the initial state.

Definition 2. *Let σ_1 and σ_2 be strategies of Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$. Strategy σ_1 is said* finer *than σ_2, denoted $\sigma_1 \ll \sigma_2$, if for every strategy σ_S of Spoiler, $d_{\mathsf{Bad}}(\mathsf{Aut}(\sigma_1, \sigma_S)) \geq d_{\mathsf{Bad}}(\mathsf{Aut}(\sigma_2, \sigma_S))$.*

Given this definition, an optimal strategy for Determinizator is a minimal element for the partial order $\ll$. Note that, if they exist, winning strategies are the optimal ones since against all strategies of Spoiler, the corresponding distance to Bad is infinite. The set of optimal strategies can be computed effectively by a fix-point computation using a rank function on the vertices of the game.

With respect to this partial order on strategies, positional strategies are sufficient for Determinizator.

Proposition 1. *For every strategy σ of Determinizator with arbitrary memory, there exists a positional strategy σ' such that $\sigma' \ll \sigma$.*

Strategy σ' is obtained from σ by letting for each state the first choice made in σ; this cannot decrease the distance to Bad. Strategies of interest for Determinizator can be even more restricted. Indeed, any timed automaton can be turned into an equivalent one with atomic resets only, using a construction similar to the one that removes clock transfers (updates of the form $x := x'$) [Bou09]. Thus, for every strategy for Determinizator there is a finer one which resets at most one clock on each transition, which can be turned into a finer positional strategy thanks to Proposition 1. As a consequence, with respect to $\ll$, positional strategies that only allow for atomic resets are sufficient for Determinizator.

4 Comparison with existing methods

The method we presented is both more precise than the algorithm of [KT09] and more general than the procedure of [BBBB09a]. Let us detail these two points.

4.1 Comparison with [KT09]

First of all, our method covers the application area of [KT09] since each time the latter algorithm produces a deterministic equivalent with resources (k, M') for a timed automaton $\mathcal{A}$, there is a winning strategy for Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$. To be more precise, in [KT09] the construction of a deterministic over-approximation is guided by a *skeleton*, a finite automaton, which governs the resets of the deterministic automaton in construction. The resets are thus defined by a regular language. Our strategies are more powerful than the skeletons of [KT09] since the resets are also determined by the regions the actions are taken in. Strategies can thus be seen as timed skeletons and the game allos us to choose a good strategy, contrary to the skeletons of [KT09] that are fixed *a priori*. Also, when a strategy is winning, we know that the determinization is exact.

Moreover, contrary to the method presented in [KT09], our game-approach is exact on deterministic timed automata: given a DTA $\mathcal{A}$ over resources (k, M), Determinizator has a winning strategy in $\mathcal{G}_{\mathcal{A},(k,M)}$. This is again a consequence of the more general fact that a strategy can be seen as a timed generalization of the notion of skeleton, and solving our game amounts to finding a relevant timed skeleton. As an example, the algorithm of [KT09] run on the timed automaton of Figure 1 produces a strict over-approximation, represented on Figure 5, while our approach is exact.

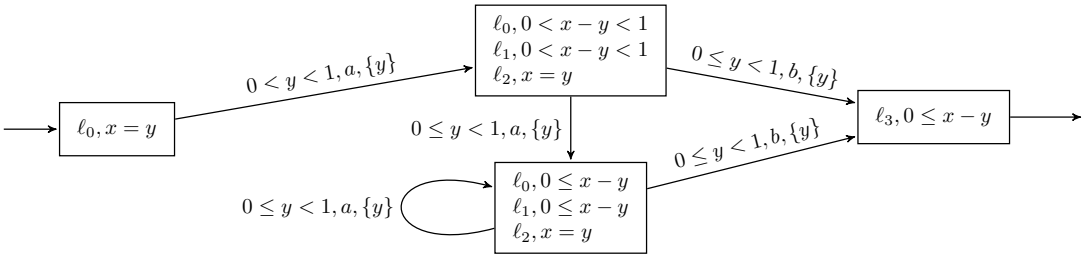

Figure 5: The result of algorithm [KT09] on the running example.

Our approach also improves the updates of the relations between clocks by taking the original guard into account. Precisely, when computing up, an intersection with the guard in the original TA is performed. This improvement allows us, even under the same resetting policy, to refine the over-approximation given by [KT09].

4.2 Comparison with [BBBB09a]

4.2.1 Generalities

Our approach generalizes the one in [BBBB09a] since, for any timed automaton $\mathcal{A}$ such that the procedure in [BBBB09a] yields an equivalent deterministic timed automaton with k clocks and maximal constant M', there is a winning strategy for Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$. This can be explained by the fact that relations between clocks of $\mathcal{A}$ and clocks in the game allow one to record more information than the mapping used in [BBBB09a]. Moreover, our approach strictly broadens the class of automata determinized by the procedure of [BBBB09a] in two respects. First of all, our method allows one to cope with some language inclusions, contrary to [BBBB09a]. For example, the TA depicted on the left-hand side of Figure 6 cannot be treated by the procedure of [BBBB09a] but is easily determinized using our approach. In this example, the language of timed words accepted in location ℓ_3 is not determinizable. This will cause the failure of [BBBB09a]. However, all timed words accepted in ℓ_3 also are accepted in ℓ_4 and the language of timed words accepted in ℓ_4 is clearly determinizable. Our approach allows one to deal with such language inclusions, and will thus provide an equivalent deterministic timed automaton. Second, the relations between clocks of the TA and clocks of the game are more precise than the mapping used in [BBBB09a], since the mapping can be seen as restricted relations: a conjunction of constraints of the form $x - y = 0$. The precision we add by considering relations rather than mappings is sometimes crucial for the determinization. For example, the TA represented on the right-hand side of Figure 6 can be determinized by our game-approach, but not by [BBBB09a].

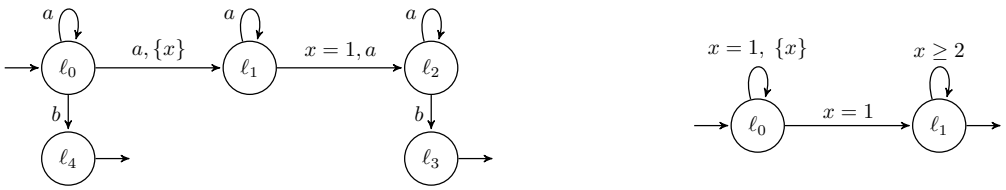

Figure 6: Examples of determinizable TA not treatable by [BBBB09a].

Apart from strictly broadening the class of timed automata that can be automatically determinized, our approach performs better on some timed automata by providing a deterministic timed automaton with less resources. This is the case on the running example of Figure 1. The deterministic automaton obtained by [BBBB09a] is depicted in Figure 7: it needs 2 clocks when our method produces a single-clock TA.

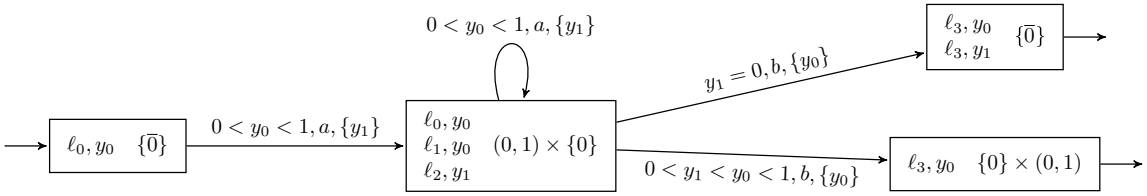

Figure 7: The result of procedure [BBBB09a] on the running example.

The same phenomenon happens with timed automata with integer resets. Timed automata with integer resets, introduced in [SPKM08], form a determinizable subclass of timed automata, where every edge (ℓ, g, a, X', ℓ') satisfies $X' \neq \emptyset$ if and only if g contains an atomic constraint of the form $x = c$ for some clock x. Intuitively, a single clock is needed to represent clocks of $\mathcal{A}$ since they all share a common fractional part.

Proposition 2. *For every timed automaton $\mathcal{A}$ with integer resets and maximal constant M, Determinizator has a winning strategy in $\mathcal{G}_{\mathcal{A},(1,M)}$.*

Proof. Let $\mathcal{A}$ be a timed automaton with integer resets over set of clocks X and maximal constant M. Note that, by definition of TA with integer resets, along any execution of $\mathcal{A}$, all clocks share a common fractional part. This very property ensures that an equivalent deterministic TA with one clock can be constructed. Precisely, in $\mathcal{G}_{\mathcal{A},(1,M)}$ we consider the strategy σ for Determinizator which resets the single clock y exactly for transitions that correspond to at least one transition of $\mathcal{A}$ containing an equality constraint (atomic constraint of the form $x = c$). Since $\mathcal{A}$ is a TA with integer resets, clocks in X cannot be reset out of these transitions. Therefore, for every clock $x \in X$, the value of y is always smaller than the one of x in $\mathsf{Aut}(\sigma)$ and each relation contains either $x - y = c$ with $0 \leq c \leq M$, or $x - y > M$. In the latter case, necessarily $x > M$. As a consequence, guards over X can always be exactly expressed in $G_M(\{y\})$. This ensures that only states where all configurations are marked $\top$ will be visited in $\mathsf{Aut}(\sigma)$. Hence, σ is winning and $\mathcal{L}(\mathsf{Aut}(\sigma)) = \mathcal{L}(\mathcal{A})$. Note that $\mathsf{Aut}(\sigma)$ is still a TA with integer resets and its size is doubly exponential in the size of $\mathcal{A}$. □

As a consequence of Proposition 2, any timed automaton with integer resets can be determinized into a doubly exponential single-clock timed automaton with the same maximal constant. This improves the result given in [BBBB09a] where any timed automaton with integer resets and maximal constant M can be turned into a doubly exponential deterministic timed automaton, using $M + 1$ clocks. Moreover, our procedure is optimal on this class thanks to the lower-bound provided in [MK10].

4.2.2 A deeper comparison with [BBBB09a]

A general comparison of our approach with [BBBB09a] is given in 4.2.1. We now discuss how our method even when restricted to equality relations (conjunctions of $x - y = c$) extends the procedure [BBBB09a].

Note that for this approach the effective construction, detailed in the technical report [BBBB09b], is similar to our approach restricted to mappings instead of relations. The use of relations, even restricted to equality relations, allows to extend several results of [BBBB09a], as we will explain here.

In order to give a precise comparison with [BBBB09a], we need to recall some elements of this paper. The authors propose two sufficient conditions for the termination of their determinization procedure: one is necessary and one is easily decidable on the region automaton. The first sufficient condition applies to some intermediary construct, $\mathsf{SymbDet}(\mathcal{R}(\mathcal{A}^\infty))$, obtained after the determinization step of the procedure. $\mathsf{SymbDet}(\mathcal{R}(\mathcal{A}^\infty))$ is an infinite tree and its nodes are labeled by a region over new clocks and a set of pairs consisting of a location of $\mathcal{A}$ together with a mapping of original clocks to new ones. In a node of this infinite tree, a clock is said *active* if it appears in one of the mappings. $\mathsf{SymbDet}(\mathcal{R}(\mathcal{A}^\infty))$ is said γ-*clock-bounded* if γ is a uniform bound on the number of active clocks in every node. A necessary and sufficient condition for termination of the procedure with γ clocks is thus that $\mathsf{SymbDet}(\mathcal{R}(\mathcal{A}^\infty))$ is γ-clock-bounded. Unfortunately, this non-syntactical condition is complex. As a consequence a stronger condition, sufficient but not necessary for termination is proposed in [BBBB09a]. Given $p \in \mathbb{N}$, a timed automaton $\mathcal{A}$ with maximal constant M and set of clocks X is said to satisfy the p-*assumption* if for every $n \geq p$, for every run $\rho = (\ell_0, v_0) \xrightarrow{\tau_1,a_1} (\ell_1, v_1) \cdots \xrightarrow{\tau_n,a_n} (\ell_n, v_n)$ in $\mathcal{A}$, for every clock $x \in X$, either x is reset along ρ or $v_n(x) > M$. Intuitively, for each clock, either it has been reset recently (within p steps), or its value is irrelevant (above the maximal constant), hence at most p new clocks are needed to represent clocks of $\mathcal{A}$.

These two sufficient conditions (γ-clock-boundedness and p-assumption) can be weakened if instead of the termination of the procedure of [BBBB09a] one is interested in the existence of a winning strategy for Determinizator in our game.

- First, if in the nodes of $\mathsf{SymbDet}(\mathcal{R}(\mathcal{A}^\infty))$, the number of different fractional parts of active clocks is bounded by γ, then Determinizator has a winning strategy in $\mathcal{G}_{\mathcal{A},(\gamma,M)}$, where M is the maximal constant of $\mathcal{A}$. Note that, compared to the clock-boundedness condition, this weakened condition is not sufficient for the termination in [BBBB09a]. Note also that it is not necessary for our approach since it only deals with timed automata such that Determinizator has a winning strategy with equality relations (conjunctions of $x - y = c$) for Determinizator.

- Second, the p-assumption can also be relaxed. Given $p \in \mathbb{N}$, a timed automaton $\mathcal{A}$ with maximal constant M and set of clocks X is said to satisfy the *weak-p-assumption* if there exists a partition $X = H \sqcup H'$ such that for every $n \geq p$, for every run $\rho = (\ell_0, v_0) \xrightarrow{\tau_1,a_1} (\ell_1, v_1) \cdots \xrightarrow{\tau_n,a_n} (\ell_n, v_n)$ in $\mathcal{A}$, for every clock $x \in H$, either x is reset along ρ or $v_n(x) > M$ and for every $m \leq n$ and every clock $x' \in H'$, there exists $h \in H$ such that $\{v_m(x')\} = \{v_m(h)\}$. In words, clocks in H satisfy the p-assumption, and clocks in H' can be expressed by the same clocks as those in H (since their fractional part agree) all along the run. Under the weak-p-assumption, it is easy to see that Determinizator can enforce to use only equality relations. As a consequence, if a timed automaton with maximal constant M satisfies the weak-p-assumption then Determinizator has a winning strategy in $\mathcal{G}_{\mathcal{A},(p,M)}$. Note that this weakened condition is not sufficient for the termination in [BBBB09a], but still decidable, by inspection of the region automaton. Note also that the weak-p-assumption naturally implies the above-mentioned weakened condition over $\mathsf{SymbDet}(\mathcal{R}(\mathcal{A}^\infty))$.

These two sufficient conditions for the existence of a winning strategy for Determinizator show that our approach generalizes the determinization procedure of [BBBB09a] even if we restrict relations in the game to equality ones.

Following the same ideas, we explain how to naturally extend the class of event-clock timed automata into a class determinizable by our game-based approach. An *event-clock* automaton over alphabet Σ is a timed automaton with one clock per action which is reset exactly when the associated action is fired. The class of event-clock timed automata is determinizable [AFH94], and can be determinized by the procedure of [BBBB09a] because their associated deterministic infinite trees are $|\Sigma|$-clock-bounded. In fact, event-clock automata enjoy an input-determinacy property: resets only depend on the timed word and not on the precise edges that are taken. To extend the class while preserving determinizability, we introduce the notions of *finitely-* and *temporally-framed* timed automata. A timed automaton $\mathcal{A}$ is finitely-framed (resp. temporally-framed) if there exists a deterministic finite automaton (resp. deterministic timed automaton) which is complete over the alphabet of actions and which governs the clocks resets in $\mathcal{A}$. These notions are inspired by the *skeleton* of Krichen and Tripakis [KT09] which is a deterministic finite automaton that

guides the construction of a deterministic over-approximation by fixing the clocks resets. A (finitely- or temporally-) framed timed automaton is determinizable by the procedure of [BBBB09a] and our game-based approach. These two characteristics can be weakened similarly as γ-clock-boundedness and p-assumption. A timed automaton $\mathcal{A}$ is *weakly finitely-framed* (resp. *weakly temporally-framed*) if there exists a partition $X = H \sqcup H'$ of its set of clocks such that there exists a complete deterministic finite automaton (resp. a complete deterministic timed automaton) which describes resets of clocks of H, and if for every clock $x' \in H'$ and every state (ℓ, v) of $\mathcal{A}$, there exists a clock $h \in H$ such that $\{v_m(x')\} = \{v_m(h)\}$. Any weakly framed automaton $\mathcal{A}$ is determinizable by our approach with resources $(|H|, M)$ where M is the maximal constant of $\mathcal{A}$, whereas the procedure [BBBB09a] would not necessarily terminate.

5 Extension to ε-transitions and invariants

In Section 3 the construction of the game and its properties were presented for a restricted class of timed automata with no ε-transitions and no invariants. Let us now explain how to extend the previous construction to deal with these two aspects.

5.1 ε-transitions

Let us first explain informally the modifications that are needed in the definition of the game to deal with ε-transitions. Quite naturally, an ε-closure has to be performed when computing new states in the game. This closure calls for an extension of the structure of the states: delays might be mandatory before taking an ε-transition, and hence, potentially distinct regions are attached to configurations of a state in the game. This phenomenon is illustrated on the example of TA depicted in Figure 8, left. The resulting game, is represented Figure 8, right, where the right-most state is the ε-closure of the former state $\{(\ell_1, x - y = 0, \top, \{0\})\}$. For instance, the configuration $(\ell_1, x - y = -2, \top)$ can only be reached after two ε-transitions of the original TA, taken respectively after one or two time units. Thus this configuration can only be observed while in region $\{2\}$ and after, whereas the configuration $(\ell_1, x-y = 0, \top)$ could be observed already in region $\{0\}$. As a consequence, within a state, configurations can have distinct associated regions, and every region is a time successor of the initial region of the state. Computing the ε-closure of a state in the game amounts to computing the set of reachable configurations

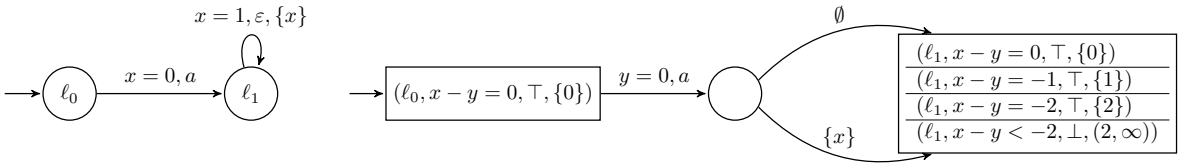

Figure 8: A timed automaton with ε-transitions and the resulting ε-closure.

by ε-transitions, and associating with every configuration its corresponding region. This computation can be seen as a construction of a branch of the game where ε would be a standard action, but where Determinizator is not allowed to reset any clock; all the states obtained this way are then gathered into a unique state. For instance, Figure 9 represents the computation of the ε-closure discussed above. This alternative point of view justifies that the computation always terminates.

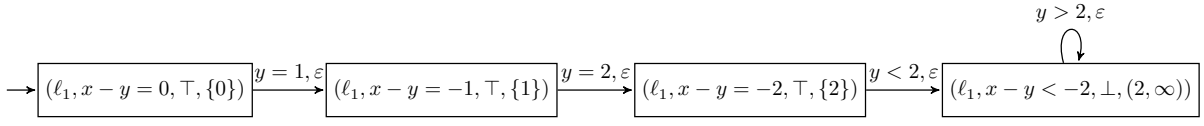

Figure 9: Step-wise computation of the ε-closure, before merging.

The set Bad also needs to be redefined when taking into account possible ε-transitions, since regions are now attached to configurations and no longer to states. More precisely, when the ε-closure leads to a configuration associated with a final location but after a nonzero delay, or when the configurations associated with a final location and the initial region (induced by the both last moves) are marked $\bot$, there is a risk of over-approximation. Therefore, when a strategy σ is fixed, these states of Spoiler become naturally final locations of $\mathsf{Aut}(\sigma)$, but Determinizator wants to avoid these states in the game and the set Bad should thus be extended.

We now come to the formal definition of the game. Given $\mathcal{A}$ and (k, M') we define $\mathcal{G}_{\mathcal{A},(k,M')} = (\mathsf{V}, \mathsf{v}_0, \mathsf{Act}, \delta, Bad)$ where:

- The set of vertices V is partitioned into V_S and V_D, respectively vertices of Spoiler and Determinizator. Vertices of V_S and V_D are labeled respectively in $2^{L \times Rel^{X,Y}_{M,M'} \times \{\top,\bot\} \times Reg^Y_{M'}}$ and $2^{L \times Rel^{X,Y}_{M,M'} \times \{\top,\bot\} \times Reg^Y_{M'}} \times (Reg^Y_{M'} \times \Sigma)$;
- $\mathsf{v}_0 = \mathsf{cl}_\varepsilon(\{(\ell_0, X - Y = 0, \top, \{\overline{0}\})\})$ is the initial vertex and belongs to player Spoiler;
- $\mathsf{Act} = (Reg^Y_{M'} \times \Sigma) \cup 2^Y$ is the set of possible actions;
- $\delta \subseteq \mathsf{V}_S \times (Reg^Y_{M'} \times \Sigma) \times \mathsf{V}_D \cup \mathsf{V}_D \times 2^Y \times \mathsf{V}_S$ is the set of edges;
- $\mathsf{Bad} = \{\{(\ell_j, C_j, \bot, r_j)\}_j\} \cup \{\{(\ell_j, C_j, b_j, r_j)\}_j \,|\, \forall k \, \big((\forall j \; r_j \in \overrightarrow{r_k}) \Rightarrow (\ell_k \in F \Rightarrow b_k = \bot)\big)\}$ is the set of bad states.

We now detail the edge relation which defines the possible moves of the players. Given a state of Spoiler $\mathsf{v}_S = \{(\ell_j, C_j, b_j, r_j)\}_j \in \mathsf{V}_S$ and (r', a) one of his moves, the successor state is defined, provided r' is a time-successor of r_j for some j, as the state $\mathsf{v}_D = (\{(\ell_j, C_j, b_j, r_j)\}_j, (r', a)) \in \mathsf{V}_D$ if there exists $(\ell, C, b, r) \in \mathsf{v}_S$, such that $r' \in \overrightarrow{r}$ and there exists $\ell \xrightarrow{g,a,X'} \ell' \in E$ with $[r' \cap C]_{|X} \cap g \neq \emptyset$.

Given $\mathsf{v}_D = (\{(\ell_j, C_j, b_j, r_j)\}_j, (r', a)) \in \mathsf{V}_D$ a state of Determinizator and $Y' \subseteq Y$ one of his moves, the successor state v_S is obtained as the ε-closure of the set of all elementary successors of configurations in $\{(\ell_j, C_j, b_j, r_j)\}_j$ by (r', a) and resetting Y'. Precisely, if (ℓ, C, b, r) is a configuration such that $r' \in \overrightarrow{r}$, its elementary successors set by (r', a) and resetting Y' is:

$$\mathsf{Succ}_e[r', a, Y'](\ell, C, b, r) = \left\{ (\ell', C', b', r'_{[Y' \leftarrow 0]}) \;\middle|\; \begin{array}{l} \exists \ell \xrightarrow{g,a,X'} \ell' \in E \text{ s.t. } [r' \cap C]_{|X} \cap g \neq \emptyset \\ C' = \mathsf{up}(r', C, g, X', Y') \\ b' = b \wedge ([r' \cap C]_{|X} \subseteq g) \end{array} \right\}$$

where $\mathsf{up}(r', C, g, X', Y')$ is the update of the relation C between clocks in X and Y after the moves of the two players, that is after taking action a in r', resetting $X' \subseteq X$ and $Y' \subseteq Y$, and forcing the satisfaction of g. Formally, $\mathsf{up}(r', C, g, X', Y') = \overleftarrow{(r' \cap C \cap g)_{[X' \leftarrow 0][Y' \leftarrow 0]}}$. Boolean b' is set to $\bot$ if either $b = \bot$ or the induced guard $[r' \cap C]_{|X}$ over-approximates g. In the update, the intersection with g aims at stopping runs that for sure will correspond to timed words out of $\mathcal{L}(\mathcal{A})$; the boolean b anyway takes care of keeping track of the possible over-approximation. To complete this definition, let us formalize the ε-closure of a state of Spoiler. The ε-closure of a configuration (ℓ, C, b, r), denoted by $\mathsf{cl}_\varepsilon(\ell, C, b, r)$, is defined as the smallest fixpoint of the functional

$$X \mapsto (\ell, C, b, r) \cup \bigcup_{\substack{(\ell', C', b', r') \in X \\ r'' \in \overrightarrow{r'}}} \mathsf{Succ}_e[r'', \varepsilon, \emptyset](\ell', C', b', r').$$

The termination of the iterative computation of the fixpoint is trivial.

5.2 Invariants

We now explain how to adapt the framework to timed automata with invariants. First, while computing the elementary successors for configurations, invariants have to be taken into account. Second, with each state of the game, we associate an invariant, taking into account the respective invariants of the original locations. Last, the set of bad states needs to be redefined.

In the computation of the successors states, the invariants are taken care of similarly to the guards: their satisfaction is checked on both extremities of the transitions. In order to do so, for a transition $\ell \xrightarrow{g,a,X'} \ell'$ the condition $[r' \cap C]_{|X} \cap g \neq \emptyset$ is replaced by $[r' \cap C]_{|X} \cap g \cap \mathsf{Inv}(\ell) \cap (\mathsf{Inv}(\ell'))_{[X' \leftarrow 0]^{-1}} \neq \emptyset$. The boolean is thus naturally defined as follows: $b' = b \wedge ([r' \cap C]_{|X} \subseteq (g \cap \mathsf{Inv}(\ell) \cap (\mathsf{Inv}(\ell'))_{[X' \leftarrow 0]^{-1}}))$. As a consequence, the configurations which are built via an approximation of some invariant are marked $\perp$. The relation updates are also redefined, to enforce the satisfaction of the invariants: $\mathsf{up}(r', C, g, \ell, \ell', X', Y') = \overleftarrow{(r' \cap C \cap g \cap \mathsf{Inv}(\ell))_{[X' \leftarrow 0][Y' \leftarrow 0]} \cap \mathsf{Inv}(\ell')}$.

To each state of Spoiler, we attach an invariant in $I_{M'}(Y)$. Moreover, since this invariant may be over-approximated (due to insufficient precision in the relation between clocks in X and Y), a boolean is associated with each invariant. Informally, a state v_S of Spoiler has the form $\mathsf{v}_S = (\{(\ell_j, C_j, b_j, r_j)\}_j, (\mathsf{I}, b))$ where I is the most restrictive invariant over Y that over-approximates every invariant for the configurations composing v_S. Formally, $\mathsf{I} = \bigcup\{r'' \in \mathsf{Reg}^Y_{M'} \mid \exists j \in J \text{ s.t. } r'' \in \overrightarrow{r_j} \wedge [r'' \cap C_j]_{|X} \cap \mathsf{Inv}(\ell_j) \neq \emptyset\}$.

The boolean b, reflecting that an over-approximation possibly happened for the invariant, is defined as $b = \bigvee_j (b_j \wedge ([\mathsf{I} \cap C_j]_{|X} \subseteq \mathsf{Inv}(\ell_j))) \wedge \bigvee_{j,\, \ell_j \in F} (b_j \wedge ([\mathsf{I} \cap C_j]_{|X} \subseteq \mathsf{Inv}(\ell_j)))$. Indeed, it is set to $\perp$ either if for every configuration some over-approximation in the invariant might have occurred, or if there is a configuration associated with a final location and for every such configuration there is a risk of over-approximation by the invariant. We now illustrate the computation of invariants on an example.

Figure 10 represents a timed automaton with invariants. One branch of the corresponding game over

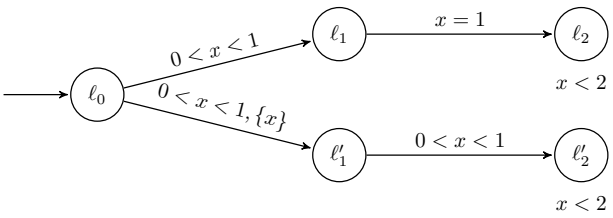

Figure 10: A timed automaton with invariants.

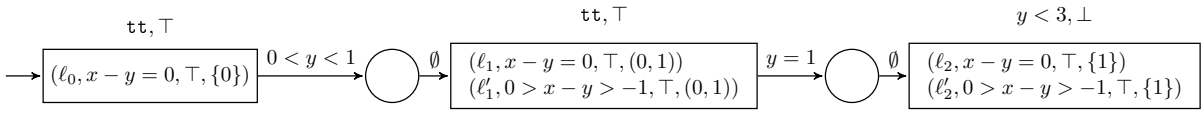

Figure 11: Branch of the game with resources $(1, 3)$.

resources $(1, 3)$ is depicted in Figure 11. The invariant corresponding to the configuration $(\ell_2, x-y = 0, \top)$ is trivially $y < 2$. The one corresponding to $(\ell'_2, 0 > x - y > -1, \top)$ is $y < 3$, since $x < 2$ implies $y < 3$, moreover the region $2 < y < 3$ is necessarily included in this invariant because *e.g.*, $y = 2.1$, $x = 1.9$ is a valuation satisfying $0 > x - y > -1$, $2 < y < 3$ and $x < 2$. Thus, the invariant associated with the state of the game is $y < 3$ and the corresponding boolean is $\perp$ because the invariant is an over-approximation.

Approximating invariants is sufficient to lead to a strict over-approximation of the original timed language. Therefore the set of bad states need to be redefined, in order to preserve that any winning strategy for Determinizator yields a deterministic equivalent to the original timed automaton. We thus define the set Bad as follows:

$$\mathsf{Bad} = \{(\{(\ell_j, C_j, \perp, r_j)\}_j, (\mathsf{I}, b))\} \cup \{(\{(\ell_j, C_j, b_j, r_j)\}_j, (\mathsf{I}, \perp))\} \cup \{(\{(\ell_j, C_j, b_j, r_j)\}_j, (\mathsf{I}, b)) \mid \forall k \left((\forall j\ r_j \in \overrightarrow{r_k}) \Rightarrow (\ell_k \in F \Rightarrow b_k = \perp)\right)\}.$$

5.3 Properties of the strategies in the extended game

Properties presented in the section 3.2 are preserved by the extension.

Theorem 2. *Let $\mathcal{A}$ a timed automaton, and $k, M' \in \mathbb{N}$. For every strategy σ of Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$, $\mathsf{Aut}(\sigma)$ is a deterministic timed automaton over resources (k, M') and satisfies $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$. Moreover, if σ is winning, then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathsf{Aut}(\sigma))$.*

Proof. The proof is split in two parts. First of all, we show that any strategy σ for Determinizator ensures that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$. Then we show that for every winning strategy σ, the reverse inclusion also holds.

($\subseteq$): Let σ be a strategy for Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$. To show that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$ we prove a stronger fact on the transition systems $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{T}_{\mathsf{Aut}(\sigma)}$. If $\mathcal{T}_{\mathcal{A}} = (S, s_0, S_F, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \rightarrow_{\mathcal{A}})$, $\mathcal{T}_{\mathsf{Aut}(\sigma)} = (S', s'_0, S'_F, (\mathbb{R}_+ \times \Sigma), \rightarrow_{\mathsf{Aut}(\sigma)})$, then the simulation relation $\rho \subseteq S \times S'$ defined as:

$$\rho = \{\big((\ell, v), ((\mathcal{E}, (\mathsf{I}, b_{\mathsf{I}})), v')\big) \mid \exists (\ell, C, b, r) \in \mathcal{E},\ (v, v') \in C\}$$

satisfies the following conditions:

1. $(s_0, s'_0) \in \rho$,
2. for all $(s, s') \in \rho$, for all $a \in \Sigma$ whenever $s \xrightarrow{\tau_1,\varepsilon}_{\mathcal{A}} s_1 \cdots \xrightarrow{\tau_{n-1},\varepsilon}_{\mathcal{A}} s_{n-1} \xrightarrow{\tau_n,a}_{\mathcal{A}} s_n$, there exists $s'_n \in S'$ such that $(s_n, s'_n) \in \rho$ and $s' \xrightarrow{\tau,a}_{\mathsf{Aut}(\sigma)} s'_n$ with $\tau = \sum_{i=1}^{n} \tau_i$.

By definition of the final locations for $\mathsf{Aut}(\sigma)$, the existence of such a relation ρ implies the language inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$. Let us thus show that ρ indeed satisfies the two conditions. The first condition about the initial states is trivially satisfied. Assume now that $s = (\ell, v)$ and $s' = ((\mathcal{E}, (\mathsf{I}, b_{\mathsf{I}})), v')$ are states of $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{T}_{\mathsf{Aut}(\sigma)}$ respectively such that $(s, s') \in \rho$. Then, there exists $(\ell, C, b, r) \in \mathcal{E}$ such that $(v, v') \in C$. Let $s \xrightarrow{\tau_1,\varepsilon}_{\mathcal{A}} s_1 \cdots \xrightarrow{\tau_{n-1},\varepsilon}_{\mathcal{A}} s_{n-1} \xrightarrow{\tau_n,a}_{\mathcal{A}} s_n$ be a sequence of transitions from s in $\mathcal{A}$. Letting $s_j = (\ell_j, v_j)$ ($j \in [1, n-1]$ and $s = s_0$), there exist edges in $\mathcal{A}$ of the form $(\ell_{j-1}, g_j, \varepsilon, X_j, \ell_j)$ with $v_{j-1} + \tau_j \models g_j \cap \mathsf{Inv}(\ell_{j-1}) \cap (\mathsf{Inv}(\ell_j))_{[X_j \leftarrow 0]^{-1}}$ and $v_j = (v_{j-1} + \tau_j)_{[X_j \leftarrow 0]}$. By definition of the ε-closure operator cl_ε, for each $j \in [0, n-1]$ there is a configuration $(\ell_j, C_j, b_j, r_j) \in \mathcal{E}$ such that $(v_j, v' + \sum_{i=1}^{j} \tau_i) \in C_j$ and $v' + \sum_{i=1}^{j} \tau_i \in r_j$. In particular, there is a configuration $(\ell_{n-1}, C_{n-1}, b_{n-1}, r_{n-1}) \in \mathcal{E}$ such that $(v_{n-1}, v' + \sum_{i=1}^{n-1} \tau_i) \in C_{n-1}$ and $v' + \sum_{i=1}^{n-1} \tau_i \in r_{n-1}$. Now, because $s_{n-1} \xrightarrow{\tau_n,a}_{\mathcal{A}} s_n$ there exists an edge in $\mathcal{A}$ of the form $(\ell_{n-1}, g_n, \varepsilon, X_n, \ell_n)$ where ℓ_n is the location of s_n. As a consequence, by definition of the game, for any region $r' \in \overrightarrow{r_{n-1}}$ and $r' \subseteq [C_{n-1} \cap g_n \cap \mathsf{Inv}(\ell_{n-1}) \cap (\mathsf{Inv}(\ell_n))_{[X_n \leftarrow 0]^{-1}}]_{|Y}$ there is an edge $(\mathcal{E}, (\mathsf{I}, b_{\mathsf{I}})) \xrightarrow{r',a,Y'} (\mathcal{E}', (\mathsf{I}', b_{\mathsf{I}'}))$ in $\mathsf{Aut}(\sigma)$ and a configuration $(\ell_n, C', b', r') \in \mathcal{E}'$ which is an elementary successor of $(\ell_{n-1}, C_{n-1}, b_{n-1}, r_{n-1})$. Letting $s'_n = ((\mathcal{E}', (\mathsf{I}', b_{\mathsf{I}'})), v'_n)$ where $v'_n = [v' + \tau]_{Y' \leftarrow 0}$, we observe that $(v_n, v'_n) \in C'$ using the definition of the updates for the relation and the fact that $(v_{n-1}, v' + \sum_{i=1}^{n-1} \tau_i) \in C_{n-1}$. Hence $(s_n, s'_n) \in \rho$. Note that the way invariants are defined for states of Spoiler (and thus locations in $\mathsf{Aut}(\sigma)$), the transition we construct in $\mathsf{Aut}(\sigma)$ satisfies the invariants at the source and target locations. This concludes the proof that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$.

($\supseteq$): Assume now that σ is a winning strategy in $\mathcal{G}_{\mathcal{A},(k,M')}$. We show that the relation ρ defined in the other direction satisfies in addition the following condition:

3. for all s', for all $a \in \Sigma$, whenever $s' \xrightarrow{\tau,a}_{\mathsf{Aut}(\sigma)} s''$ there exists s such that $(s, s') \in \rho$ and there exists $s \xrightarrow{\tau_1,\varepsilon}_{\mathcal{A}} s_1 \cdots \xrightarrow{\tau_{n-1},\varepsilon}_{\mathcal{A}} s_{n-1} \xrightarrow{\tau_n,a}_{\mathcal{A}} s_n$ with $\sum_{i=1}^{n} \tau_i = \tau$ and $(s_n, s'') \in \rho$.

Again, because of the definition of final locations in $\mathsf{Aut}(\sigma)$ and the fact that σ is winning, the third condition on ρ implies that $\mathcal{L}(\mathsf{Aut}(\sigma)) \subseteq \mathcal{L}(\mathcal{A})$. This relies on two observations: first of all the definition of Bad ensures that safe locations are never partially accepting (that is, non-accepting for some region, and then accepting for a time-successor); second the boolean associated with the invariant is always $\top$, hence in locations of $\mathsf{Aut}(\sigma)$ the invariant is never over-approximated for some final configuration. Let us now show that ρ satisfies the last condition. Let $s' = ((\mathcal{E}', (\mathsf{I}', \top)), v')$ be a state of $\mathsf{Aut}(\sigma)$ and $s' \xrightarrow{\tau,a}_{\mathsf{Aut}(\sigma)} s''$. Hence, there is an edge $(\mathcal{E}', (\mathsf{I}', \top)) \xrightarrow{r',a,Y'} (\mathcal{E}'', (\mathsf{I}'', \top))$ in $\mathsf{Aut}(\sigma)$ and $s'' = ((\mathcal{E}'', (\mathsf{I}'', \top)), v'')$. Since σ is winning, there must be a configuration $(\ell_n, C_n, b_n, r_n) \in \mathcal{E}''$ with $b_n = \top$. Let v_n be any valuation such that $(v_n, v'') \in C_n$ (such a v_n exists since $v'' \models [C_n \cap r_n]_{|Y}$). Thus the pair (s_n, s'') where $s_n = (\ell_n, v_n)$ belongs to ρ. The configuration $(\ell_n, C_n, \top, r_n)$ has a predecessor in $\mathcal{E}'$ by action a, say $(\ell_{n-1}, C_{n-1}, b_{n-1}, r_{n-1})$ and b_{n-1} is necessarily $\top$ otherwise b_n would also be $\bot$. Let $(\ell_{n-1}, g_n, a, X_n, \ell_n)$ be the edge in $\mathcal{A}$ that generated the construction of $(\ell_n, C_n, \top, r_n)$ from $(\ell_{n-1}, C_{n-1}, \top, r_{n-1})$. Let $\tilde{v}_{n-1} \in [C_{n-1} \cap (v_n)_{[X_n \leftarrow 0]^{-1}}]_{|X}$ and $\tilde{s}_{n-1} = (\ell_{n-1}, \tilde{v}_{n-1}) \xrightarrow{0,a}_{\mathcal{A}} s_n$ with $(\tilde{v}_{n-1}, v' + \tau) \in C_{n-1}$.

Case 1 If $v' \in r_{n-1}$, we let $v_{n-1} = \tilde{v}_{n-1} - \tau$. Then $(v_{n-1}, v') \in C_{n-1}$, there is a transition $s_{n-1} = (\ell_{n-1}, v_{n-1}) \xrightarrow{\tau,a}_{\mathcal{A}} s_n$ and $(s_{n-1}, s') \in \rho$. In this case, we can conclude.

Case 2 If $v' \notin r_{n-1}$, we let r' such that $v' \in r'$. Then $r_{n-1} \in \overrightarrow{r'}$ and we define τ_n as the maximal delay such that $v'+(\tau-\tau_n) \in r_{n-1}$. Observe that $(v_{n-1}, v'+(\tau-\tau_n)) \in C_{n-1}$ where $v_{n-1} = \tilde{v}_{n-1}-\tau_n$. The situation for $(s_{n-1}, s'+(\tau-\tau_n)) \in \rho$ is now similar to the one for (s_n, s'') since $v'+(\tau-\tau_n) \in r_{n-1}$ (as $v' \in r_n$). Therefore, the configuration $(\ell_{n-1}, C_{n-1}, \top, r_{n-1})$ can be lifted to a configuration $(\ell_{n-2}, C_{n-2}, \top, r_{n-2}) \in \mathcal{E}'$ which is an elementary predecessor by an ε-transition. This process is iterated until some configuration $(\ell, C, \top, r) \in \mathcal{E}'$ is reached with $r = r'$. At this point, some state s of $\mathcal{A}$ can be defined with $(s, s') \in \rho$ and there exists a run $s \xrightarrow{\tau_1,\varepsilon}_{\mathcal{A}} s_1 \cdots \xrightarrow{\tau_{n-1},\varepsilon}_{\mathcal{A}} s_{n-1}$; the τ_i's are built the same way as τ_n.

This proves that ρ satisfies the third condition. As a consequence $\mathcal{L}(\mathsf{Aut}(\sigma)) \subseteq \mathcal{L}(\mathcal{A})$. □

5.4 Comparison with [KT09]

In Section 4, we compared our approach with existing methods in the restricted case where timed automata have neither invariants nor ε-transitions. The determinization procedure of [BBBB09a] does not deal with invariants and ε-transitions. We therefore compare here our extended approach only with the algorithm of [KT09]. The model in [KT09] consists of timed automata with silent transitions and actions are classified with respect to their urgency: eager, lazy or delayable. First of all, the authors propose an ε-closure computation which does not terminate in general, and bring up the fact that termination can be ensured by some abstraction. Second, the urgency in the model is not preserved by their over-approximation construction which only produces lazy transitions (since lazy is the over-approximation of all kinds of urgency). Note that we classically decided to use invariants to model urgency, but our approach could be adapted to the same model as they use, while preserving urgency much more often, the same way as we do for invariants. These observations underline the benefits of our game-based approach extended to deal with invariants and ε-transitions compared to existing work.

6 Other useful approximations

We presented a game-based approach which yields a deterministic over-approximation. Depending on the context, under-approximations might be more suitable than over-approximations. We therefore explain in this section how to adapt the framework to generate deterministic under-approximations, and also combine over- and under-approximations.

6.1 Under-approximation

For over-approximations, during the construction of the game, all litigious successors (those marked $\perp$) are built, possibly introducing more behaviors than in the original TA. In order to yield an under-approximation, the litigious successors are simply not constructed. The general case with ε-transitions and invariants is also easy to handle: (1) the ε-closure is under-approximated in the same way; (2) the invariant of a state is redefined as the union of all regions such that the induced guard is included in the invariant of the location of some configuration marked $\top$; and (3) finally, the set Bad is defined as the set of states where either some litigious successor existed (but was not built), or for which the invariant or the ε-closure has been under-approximated.

A motivation for building deterministic under-approximations is that the language inclusion is decidable when the 'largest' language is recognized by a deterministic timed automaton. Therefore, given $\mathcal{A}$ a timed automaton, for every deterministic under-approximation $\mathcal{B}$, one can decide whether the approximation is strict or not, that is whether $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. As a consequence, one can detect if a losing strategy yet yields a deterministic equivalent to the original timed automaton.

6.2 Combining over- and under-approximation

Combining over- and under-approximations might sometimes be useful. For example, in the context of testing, preserving the conformance relation amounts to over-approximate output actions and

under-approximate input actions. We now explain more generally how to combine over- and under-approximations. To this aim, we consider timed automata with a partitioned alphabet, $\Sigma = \Sigma_1 \sqcup \Sigma_2$, and introduce the notion of (Σ_1, Σ_2)-refinement relation[3].

Definition 3. *Let $\mathcal{A}$ be a TA and $\mathcal{A}'$ be a DTA over the alphabet $\Sigma = \Sigma_1 \sqcup \Sigma_2$, and $\mathcal{T}_{\mathcal{A}} = (S, s_0, S_F, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \rightarrow_{\mathcal{A}})$, $\mathcal{T}_{\mathcal{A}'} = (S', s'_0, S'_F, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \rightarrow_{\mathcal{A}'})$ their associated transition systems. We say that $\mathcal{A}$ (Σ_1, Σ_2)-refines $\mathcal{A}'$ and write $\mathcal{A} \preceq \mathcal{A}'$ if there exists a relation $\rho \subseteq S \times S'$ such that:*

- $(s_0, s'_0) \in \rho$;
- *if $(s, s') \in \rho$ and $s \xrightarrow{\tau_1,\varepsilon}_{\mathcal{A}} s_1 \cdots \xrightarrow{\tau_{k-1},\varepsilon}_{\mathcal{A}} s_{k-1} \xrightarrow{\tau_k,a_2}_{\mathcal{A}} \tilde{s}$ with $a_2 \in \Sigma_2$ then there exists $\tilde{s}' \in S'$ such that $s' \xrightarrow{\sum_{i=1}^k \tau_i, a_2}_{\mathcal{A}'} \tilde{s}'$ and $(\tilde{s}, \tilde{s}') \in \rho$;*
- *for all $s' \in S'$, if $s' \xrightarrow{\tau,a_1}_{\mathcal{A}'} \tilde{s}'$ with $a_1 \in \Sigma_1$ then there exists $s, \tilde{s} \in S$ such that $(s, s') \in \rho$, $(\tilde{s}, \tilde{s}') \in \rho$, and $s \xrightarrow{\tau_1,\varepsilon}_{\mathcal{A}} s_1 \cdots \xrightarrow{\tau_{k-1},\varepsilon}_{\mathcal{A}} s_{k-1} \xrightarrow{\tau_k,a_1}_{\mathcal{A}} \tilde{s}$ where $\sum_{i=1}^k \tau_i = \tau$.*

In particular, if Σ_1 and Σ_2 consist respectively in the input and output alphabets, the (Σ_1, Σ_2)-refinement relation generalizes the io-refinement relation between deterministic timed automata introduced in [DLL+10], which itself generalizes the alternating simulation [AHKV98]. Moreover, the inverse relation (generalized io-abstraction) still preserves the **tioco** conformance relation [KT09]: implementations that conform to a specification also conform to any io-abstraction of this specification. As a consequence soundness of test cases is preserved by io-refinement: a test suite sound for a given specification is also sound for any io-refinement of the specification.

The aim is thus to combine over- and under-approximations in the construction of the game such that any strategy for Determinizator yields a deterministic (Σ_1, Σ_2)-abstraction of the original automaton. The construction is therefore adapted by over-approximating transitions over actions of Σ_2 as well as invariants and under-approximating transitions over actions of Σ_1. To take into account possible ε-transitions, the structure of the states of Spoiler needs to be enriched. Indeed the ε-closure should be over-approximated before a Σ_2-action and under-approximated before a Σ_1-action. As a consequence, the set of configurations is replaced by a pair of sets which are respectively over- and under-approximated.

Formally, given $\mathcal{A}$ a timed automaton and (k, M') resources we define $\mathcal{G}_{\mathcal{A},(k,M')} = (\mathsf{V}, \mathsf{v}_0, \mathsf{Act}, \delta, \mathsf{Bad})$ where:

- Vertices V are partitioned into $\mathsf{V}_S \sqcup \mathsf{V}_D$, and vertices of V_S and V_D are labeled respectively in $(2^{L \times \mathsf{Rel}_{M,M'}(X,Y) \times \{\top,\bot\} \times \mathsf{Reg}^Y_{M'}})^2 \times (I_M(X) \times \{\top, \bot\})$ and $2^{L \times \mathsf{Rel}_{M,M'}(X,Y) \times \{\top,\bot\} \times \mathsf{Reg}^Y_{M'}} \times (\mathsf{Reg}^Y_{M'} \times \Sigma)$;
- $\mathsf{v}_0 = ((\{(\ell_0, X - Y = 0, \top, \{\overline{0}\})\}, \{(\ell_0, X - Y = 0, \top, \{\overline{0}\})\}), (\mathsf{Inv}(\ell_0)_{[X \leftarrow Y]}{}^4, \top))$ is the initial vertex and belongs to player Spoiler;
- $\mathsf{Act} = (\mathsf{Reg}^Y_{M'} \times \Sigma) \cup 2^Y$ is the set of possible actions;
- $\delta \subseteq \mathsf{V}_S \times (\mathsf{Reg}^Y_{M'} \times \Sigma) \times \mathsf{V}_D \cup \mathsf{V}_D \times 2^Y \times \mathsf{V}_S$ is the set of edges;
- $\mathsf{Bad} = \{(((\{(\ell_j, C_j, \bot, r_j)\}_j, \mathcal{E}^-), (\mathsf{I}, b))\} \cup \{((\mathcal{E}^+, \mathcal{E}^-), (\mathsf{I}, \bot))\}$
  $\cup \{(((\{(\ell_j, C_j, b_j, r_j)\}_j, \mathcal{E}^-), (\mathsf{I}, b)) \mid \forall k \left((\forall j\, r_j \in \overrightarrow{r_k}) \Rightarrow (\ell_k \in F \Rightarrow b_k = \bot)\right)\}$
  $\cup \{((\mathcal{E}^+, \mathcal{E}^-), (\mathsf{I}, b)) \mid \exists s \in \mathcal{E}^-,\ a \in \Sigma_1,\ r' \text{ and } Y' \text{ s.t. } \mathsf{Succ}^-_e[r', a, Y'](s) \neq \mathsf{Succ}^+_e[r', a, Y'](s)\}$
  is the set of bad states.

The possible moves of the players are then defined as follows. Given $\mathsf{v}_S = ((\mathcal{E}^+, \mathcal{E}^-), (\mathsf{I}, b)) \in \mathsf{V}_S$ a state of Spoiler and (r', a) one of his moves, the successor state is defined as a state $\mathsf{v}_D = (\mathcal{E}, (r', a)) \in \mathsf{V}_D$. The definition depends on a: if $a \in \Sigma_2$ we perform an over-approximation, therefore $\mathcal{E} = \mathcal{E}^+$. Note that v_D is only built if $\exists (\ell, C, b, r) \in \mathcal{E}^+,\ r' \in \overrightarrow{r}$ and $\exists \ell \xrightarrow{g,a,X'} \ell' \in E$ s.t. $[r' \cap C]_{|X} \cap g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}} \neq \emptyset$. Otherwise (if $a \in \Sigma_1$) we perform an under-approximation, therefore $\mathcal{E} = \mathcal{E}^-$. Here again, v_D is built only if $\exists ((\ell, C, b), r) \in \mathcal{E}^-,\ r' \in \overrightarrow{r}$ and $\exists \ell \xrightarrow{g,a,X'} \ell' \in E$ s.t. $[r' \cap C]_{|X} \subseteq g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}}$.

[3] Note that (Σ_1, Σ_2)-refinement is only defined here for a pair of timed automata $(\mathcal{A}, \mathcal{A}')$ when $\mathcal{A}'$ is deterministic.

[4] $\mathsf{Inv}(\ell_0)_{[X \leftarrow Y]}$ is a shortcut to denote $[\mathsf{Inv}(\ell_0) \cap X - Y = 0]_{|Y}$ the invariant over Y naturally induced by $\mathsf{Inv}(\ell_0)$ over X and $X - Y = 0$.

Given $\mathsf{v}_D = (\mathcal{E}, (r', a)) \in \mathsf{V}_D$ a state of Determinizator and $Y' \subseteq Y$ one of his moves, the successor state v_S is obtained by over- and under-approximated ε-closures of the set of elementary successors of configurations in $\mathcal{E}$ by (r', a) and resetting Y' together with the corresponding invariant and boolean. Given (ℓ, C, b, r) a configuration such that r' is a time-successor of r we detail the computation of elementary successors depending on a. If $a \in \Sigma_2$, its elementary successors set by (r', a) and Y' is:

$$\mathsf{Succ}_e^+[r', a, Y'](\ell, C, b, r) = \left\{ (\ell', C', b', r'_{[Y' \leftarrow 0]}) \;\middle|\; \begin{array}{l} \exists \ell \xrightarrow{g,a,X'} \ell' \in E \text{ s.t. } [r' \cap C]_{|X} \cap g \neq \emptyset \\ C' = \mathsf{up}(r', C, g, \mathsf{Inv}(\ell), \mathsf{Inv}(\ell'), X', Y') \\ b' = b \wedge ([r' \cap C]_{|X} \subseteq g) \end{array} \right\}$$

Now, if $a \in \Sigma_1$, its elementary successors set by (r', a) and Y' is:

$$\mathsf{Succ}_e^-[r', a, Y'](\ell, C, b, r) = \left\{ (\ell', C', b', r'_{[Y' \leftarrow 0]}) \;\middle|\; \begin{array}{l} \exists \ell \xrightarrow{g,a,X'} \ell' \in E \text{ s.t.} \\ [r' \cap C]_{|X} \subseteq g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}} \\ C' = \mathsf{up}(r', C, g, \mathsf{Inv}(\ell), \mathsf{Inv}(\ell'), X', Y') \\ b' = b \end{array} \right\}$$

In both definitions, $\mathsf{up}(r', C, g, \mathsf{Inv}(\ell), \mathsf{Inv}(\ell'), X', Y')$ is the update of the relation C between clocks in X and Y after the moves of the two players, that is after taking action a in r', resetting $X' \subseteq X$ and $Y' \subseteq Y$, and forcing the satisfaction of g, $\mathsf{Inv}(\ell)$ and $\mathsf{Inv}(\ell')$. Formally, $\mathsf{up}(r', C, g, X', Y') = \overleftarrow{(r' \cap C \cap g \cap \mathsf{Inv}(\ell))_{[X' \leftarrow 0][Y' \leftarrow 0]} \cap \mathsf{Inv}(\ell')}$.

To formalize over- and under-approximated ε-closures of a set of configurations, we define ε-closures of a single configuration. The closure of a set of configurations being the union of the closures of the individual configurations. Given (ℓ, C, b, r) a configuration, its ε-closures noted $\mathsf{cl}_\varepsilon^+(\ell, C, b, r)$ and $\mathsf{cl}_\varepsilon^-(\ell, C, b, r)$, are the smallest fixpoints of the functionals

$$X \mapsto (\ell, C, b, r) \cup \bigcup_{\substack{(\ell', C', b', r') \in X \\ r'' \in \overrightarrow{r'}}} \mathsf{Succ}_e^+[r'', \varepsilon, \emptyset](\ell', C', b', r'), \text{ and}$$

$$X \mapsto (\ell, C, b, r) \cup \bigcup_{\substack{(\ell', C', b', r') \in X \\ r'' \in \overrightarrow{r'}}} \mathsf{Succ}_e^-[r'', \varepsilon, \emptyset](\ell', C', b', r'), \text{respectively.}$$

The termination of the iterative computations of both fixpoints is trivial.

To complete the definition of successors, let us discuss the definition of the invariants associated with the states of the game. Similarly to the over-approximation case, invariants are over-approximated. This is no problem concerning under-approximations, since guards always are intersected with the original invariants, rather than the approximated one, in the construction of the game. However, under-approximating invariants could hinder over-approximations by constraining to much the guards. As in the previous section, the invariant I of a state $\mathsf{v}_S = ((\{(\ell_j, C_j, b_j, r_j)\}_j, \mathcal{E}^-), (\mathsf{I}, b)) \in \mathsf{V}_S$ is defined as follows: $\mathsf{I} = \bigcup_j r_j \cup \{r' \in \mathsf{Reg}^Y_{M'} \mid \exists j \; r' \in \overrightarrow{r_j} \wedge [r' \cap C_j]_{|X} \cap \mathsf{Inv}(\ell_j) \neq \emptyset\}$. Intuitively, I is the most restrictive invariant over Y that over-approximates all invariants corresponding to a configuration in the over-approximation part ($\mathcal{E}^+$) of v_S. If the invariant is an over-approximation for each configuration, there is a risk of over-approximation of the original language. Another source of possible over-approximations of the original language is when the invariant I over-approximates all invariants corresponding to a final location. The potential over-approximations due to the invariant are reflected in the boolean b, thus defined by $b = \bigvee_j (b_j \wedge ([\mathsf{I} \cap C_j]_{|X} \subseteq \mathsf{Inv}(\ell_j))) \wedge \bigvee_{j,\, \ell_j \in F} (b_j \wedge ([\mathsf{I} \cap C_j]_{|X} \subseteq \mathsf{Inv}(\ell_j)))$.

Last, in order to preserve exactness of winning strategies for Determinizator, the set Bad has been extended: states obtained by a strict under-approximation are added to Bad. More precisely, any state of Spoiler v_S containing a configuration (ℓ, C, b, r) such that for (r', a_1) and Y' moves of the two players $\mathsf{Succ}_e^+[r', a_1, Y'](\ell, C, b, r)$ and $\mathsf{Succ}_e^-[r', a_1, Y'](\ell, C, b, r)$ differ, is in Bad.

This extension is necessary, as demonstrated on a simple example represented in Figure 12. The strategy for Determinizator consisting in resetting y at each transition would be winning without the extension. However, this strategy clearly yields a strict under-approximation (see the corresponding automaton in Figure 13).

Under all these modifications of the game, the following proposition holds:

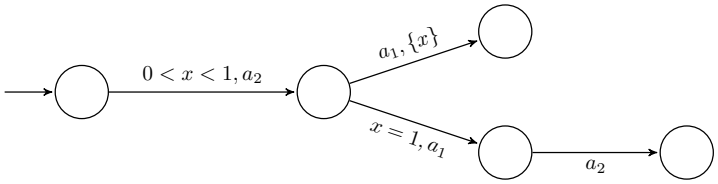

Figure 12: Example of a TA illustrating the need to extend Bad.

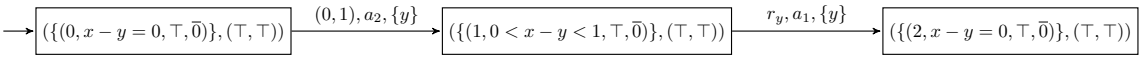

Figure 13: Automaton for a winning strategy for Determinizator corresponding to a strict under-approximation in the case where Bad is not extended.

Proposition 3. *Let $\mathcal{A}$ be a TA over the alphabet $\Sigma = \Sigma_1 \sqcup \Sigma_2$, and $k, M' \in \mathbb{N}$. For every strategy σ of Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$, $\mathsf{Aut}(\sigma)$ is a deterministic TA over resources (k, M') and $\mathcal{A} \preceq \mathsf{Aut}(\sigma)$. Moreover, if σ is winning, then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathsf{Aut}(\sigma))$.*

Proof. By definition of the game, any strategy σ yields a deterministic timed automaton $\mathsf{Aut}(\sigma)$ over resources (k, M'). The difficult part of the proof concerns arbitrary strategies. Assuming σ is a strategy for Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$, let us prove that $\mathsf{Aut}(\sigma)$ is a (Σ_1, Σ_2)-abstraction of $\mathcal{A}$, that is $\mathcal{A} \preceq \mathsf{Aut}(\sigma)$. To do so, we exhibit a relation ρ between states of $\mathcal{A}$ and $\mathsf{Aut}(\sigma)$ satisfying the three conditions of (Σ_1, Σ_2)-refinement:

$$\rho = \{\Big((\ell, v), \big(((\mathcal{E}^+, \mathcal{E}^-), (\mathsf{I}, b_{\mathsf{I}})), \tilde{v}\big)\Big) \mid \exists (\ell, C, b, r) \in \mathcal{E}^+,\ (v, \tilde{v}) \in C\}.$$

Note that states of $\mathsf{Aut}(\sigma)$ consist of a state of Spoiler in the game and a valuation.

- The first condition concerns the initial states and is trivially satisfied.
- Intuitively, the second condition holds because Σ_2-actions are over-approximated in the construction of the game. More precisely, let $(s, s') \in \rho$ with $s = (\ell_s, v_s)$ and $s' = (((\mathcal{E}^+_{s'}, \mathcal{E}^-_{s'}), (\mathsf{I}_{s'}, b_{s'})), v_{s'}))$. For every sequence of transitions $s \xrightarrow{\tau_1,\varepsilon}_{\mathcal{A}} s_1 \cdots \xrightarrow{\tau_{k-1},\varepsilon}_{\mathcal{A}} s_{k-1} \xrightarrow{\tau_k,a_2}_{\mathcal{A}} \tilde{s}$ with $a_2 \in \Sigma_2$, we denote $s_j = (\ell_j, v_j)$, for $1 \leq j \leq k-1$. By definition of the over-approximated ε-closure $\mathsf{cl}^+_\varepsilon$,for all j, there exists a configuration (ℓ_j, C_j, b_j, r_j) in $\mathcal{E}^+_{s'}$ such that $(v_j, v_{s'} + \sum_{i=1}^{j} \tau_i)$ satisfies the relation C_j. The definition of ρ thus ensures that the pair (s_{k-1}, s'') belongs to ρ, where $s'' = (((\mathcal{E}^+_{s'}, \mathcal{E}^-_{s'}), (\mathsf{I}_{s'}, b_{s'})), v_{s'} + \sum_{i=1}^{k-1} \tau_i))$. Using the definition of the successors by a Σ_2-action, there exists $\tilde{s}'$ such that $s'' \xrightarrow{\tau_k,a_2}_{\mathsf{Aut}(\sigma)} \tilde{s}'$, $(\tilde{s}, \tilde{s}') \in \rho$ and therefore $\tilde{s} \xrightarrow{\sum_{i=1}^{k} \tau_i, a_2}_{\mathsf{Aut}(\sigma)} \tilde{s}'$.
- Last, the third condition is satisfied since Σ_1-actions are under-approximated in the game construction. More precisely, let s' be a state of $\mathcal{T}_{\mathsf{Aut}(\sigma)}$ and $s' \xrightarrow{\tau,a_1}_{\mathsf{Aut}(\sigma)} \tilde{s}'$ with $a_1 \in \Sigma_1$. We write $s' + \tau$ for $(((\mathcal{E}^+_{s'}, \mathcal{E}^-_{s'}), (\mathsf{I}_{p'}, b_{p'})), v_{s'} + \tau)$. Because Σ_1-actions are under-approximated, there exists s'' and s' such that $(s'', s' + \tau) \in \rho$, $s'' \xrightarrow{0,a_1}_{\mathcal{A}} \tilde{s}$ and $(\tilde{s}, \tilde{s}') \in \rho$. Moreover, by definition of the under-approximated ε-closure $\mathsf{cl}^-_\varepsilon$, there exists $s \xrightarrow{\tau_1,\varepsilon}_{\mathcal{A}} s_1 \cdots \xrightarrow{\tau_{k-1},\varepsilon}_{\mathcal{A}} s_{k-1} \xrightarrow{\tau_k,\varepsilon}_{\mathcal{A}} s''$ such that for all $0 \leq j \leq k-1$ the pair $(s_j, (((\mathcal{E}^+_{s'}, \mathcal{E}^-_{s'}), (\mathsf{I}_{s'}, b_{s'})), v_{s'} + \sum_{i=1}^{j} \tau_i))$ belongs to ρ. In particular, $(s, s') \in \rho$ and therefore the third condition is met.

As a consequence, the above-defined relation ρ satisfies the three conditions of the definition of (Σ_1, Σ_2)-refinement and thus $\mathcal{A} \preceq \mathcal{B}$.

Assume now that σ is winning. Thanks to the extended definition of Bad, in this case we recover the properties of the original method. Indeed, for all locations of $\mathsf{Aut}(\sigma)$ there is certainly no under-approximation, due to the condition $\mathsf{Succ}^+_e[r', a_1, Y'](\ell, C, b, r) = \mathsf{Succ}^-_e[r', a_1, Y'](\ell, C, b, r)$. Therefore $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. □

Note that this extension to under- and over-approximation could be more precise provided we use another boolean to track under-approximations. Following this idea, we could still build the litigious successors while marking under-approximation and avoid to build states for which all configurations are marked. This method is *a priori* less pessimistic and would yield more precise approximations. As a drawback, the definition would be even heavier.

7 Conclusion

Contribution In this report, we proposed a game-based approach for the determinization of timed automata. Given a timed automaton $\mathcal{A}$ (with ε-transitions and invariants) and resources (k, M), we build a finite turn-based safety game between two players Spoiler and Determinizator, such that any strategy for Determinizator yields a deterministic over-approximation of the language of $\mathcal{A}$ and any winning strategy provides a deterministic equivalent for $\mathcal{A}$. We also detail how to adapt the framework to generate deterministic under-approximations, or even deterministic approximations combining under- and over-approximations. The motivation for this generalization is to tackle the problem of off-line model-based test generation from non-deterministic timed automata specifications.

Comparison with related work Our construction strictly covers and improves two existing approaches [KT09, BBBB09a]. In comparison with [KT09], our game approach yields much more often a deterministic equivalent than their over-approximation algorithm. In particular, our approach preserves deterministic timed automata (when sufficient resources are provided), which is not the case for [KT09]. This comes from the fact that our strategies can be seen as a generalization of the skeletons of [KT09]: strategies are timed and adaptative (compared to fixed finite-state skeletons). Another interesting point is that our method deals with urgency in a finer way, preserving as much as possible the invariants, whereas the algorithm of [KT09] always over-approximates the urgency status of the transitions as lazy. Compared to the determinization procedure of [BBBB09a], our approach deals with a richer model of timed automata, including ε-transitions and invariants. Also, even without these extensions, any timed automaton that can be determinized by [BBBB09a], can also be determinized by our game-approach. The class of determinized timed automata is stricly increased, thanks to a smoother treatment of relations between the original and the new clocks, and also due to a partial treatment of languages inclusions between distinct branches of the original automaton. As a consequence, the criteria given in [BBBB09a] for the termination of their abstract procedure can be weakened into sufficient conditions for the existence of a winning strategy (hence an exact deterministic equivalent) in our game.

References

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

[AFH94] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. A determinizable class of timed automata. In *Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1994.

[AHKV98] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR '98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1998.

[AMPS98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proceedings of the 5th IFAC Symposium on System Structure and Control (SSSC'98)*, pages 469–474. Elsevier Science, 1998.

[BBBB09a] Christel Baier, Nathalie Bertrand, Patricia Bouyer, and Thomas Brihaye. When are timed automata determinizable? In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP'09)*, volume 5556 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2009.

[BBBB09b] Christel Baier, Nathalie Bertrand, Patricia Bouyer, and Thomas Brihaye. When are timed automata determinizable? Research Report LSV-09-08, Laboratoire Spécification et Vérification, ENS Cachan, France, April 2009. 32 pages.

[BCD05] Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza. Fault diagnosis using timed automata. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2005.

[Bou09] Patricia Bouyer. *From Qualitative to Quantitative Analysis of Timed Systems*. Mémoire d'habilitation, Université Paris 7, Paris, France, January 2009.

[DLL+10] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Timed i/o automata: a complete specification theory for real-time systems. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC'10)*, pages 91–100. ACM, 2010.

[Fin06] Olivier Finkel. Undecidable problems about timed automata. In *Proceedings of the 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2006.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[KT09] Moez Krichen and Stavros Tripakis. Conformance testing for real-time systems. *Formal Methods in System Design*, 34(3):238–304, 2009.

[MK10] Lakshmi Manasa and Shankara Narayanan Krishna. Integer reset timed automata: Clock reduction and determinizability. CoRR arXiv:1001.1215v1, 2010.

[SPKM08] P. Vijay Suman, Paritosh K. Pandya, Shankara Narayanan Krishna, and Lakshmi Manasa. Timed automata with integer resets: Language inclusion and expressiveness. In *Proceedings of the 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, volume 5215 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2008.

[Tri06] Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. *Information Processing Letters*, 99(6):222–226, 2006.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399