A Game Approach to Determinize Timed Automata

Nathalie Bertrand[1], Amélie Stainer[1], Thierry Jéron[1], and Moez Krichen[2]

[1] INRIA Rennes - Bretagne Atlantique, Rennes, France
[2] Institute of Computer Science and Multimedia, Sfax, Tunisia

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 with ε-transitions and invariants, 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 [4] and more precise than the approximation algorithm of [11].

1 Introduction

Timed automata (TA), introduced in [1], 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 [1]. 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 [9,14]. 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 [3], event-clock TAs [2], or TAs with integer resets [13]. In a recent paper, Baier, Bertrand, Bouyer and Brihaye [4] propose a procedure which does not terminate in general, but allows one to determinize TAs in a class covering all the aforementioned determinizable classes.

M. Hofmann (Ed.): FOSSACS 2011, LNCS 6604, pp. 245–259, 2011.

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 [11]. 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.

Our method combines techniques from [4] and [11] and improves those two approaches, despite their notable differences. Moreover, it deals with both invariants and ε-transitions, but for clarity we present these treatments as extensions. Our method is also inspired by a game approach to decide the diagnosability of TAs with fixed resources presented by Bouyer, Chevalier and D'Souza in [8]. Similarly to [11], 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 [4], 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. Contrary to [4] our techniques apply to a larger class of timed automata: TAs with ε-transitions and invariants. It is also more precise than the algorithm of [11] 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 [11] 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.

The structure of the paper 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. Extensions of the method to timed automata with invariants and ε-transitions are then presented in Section 4. A comparison with existing methods is detailed in Section 5.

Due to space limitation, most proofs are omitted in this paper. All details can be found in the research report [6].

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 [1].

Given a finite set of clocks X, a clock *valuation* is a mapping $v : X \to \mathbb{R}_{\geq 0}$. We note $\overline{0}$ the valuation that assigns 0 to all clocks. If v is a valuation over X and $t \in \mathbb{R}_{\geq 0}$, then $v+t$ denotes the valuation which assigns to every clock $x \in X$ the value $v(x)+t$, and $\overleftrightarrow{v} = \{v+t \mid t \in \mathbb{R}\}$ 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', and $v_{|X'}$ for the valuation v restricted to X'.

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) 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 $\mathcal{T}_\mathcal{A} = (S, s_0, S_F, (\mathbb{R}_{\geq 0} \times (\Sigma \cup \{\varepsilon\})), \to)$ where $S = L \times \mathbb{R}_{\geq 0}^X$ is the set of states, $s_0 = (\ell_0, \overline{0})$ the initial state, $S_F = F \times \mathbb{R}_{\geq 0}^X$ the final states, and $\to \subseteq S \times (\mathbb{R}_{\geq 0} \times (\Sigma \cup \{\varepsilon\})) \times S$ 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 $(\mathbb{R}_{\geq 0} \times \Sigma)^*$ such that $(t_i)_{i \leq n}$ is nondecreasing. The timed word associated with ρ is $w = (t_{i_1}, a_{i_1}) \dots (t_{i_m}, a_{i_m})$ where $(a_i \in$

Σ 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 [1]; moreover, the determinizability of timed automata is an undecidable problem, even with fixed resources [14,9].

An example of a timed automaton is depicted in Figure 1. This nondeterministic 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 following language: $\mathcal{L}(\mathcal{A}) = \{(t_1, a) \cdots (t_n, a)(t_{n+1}, b) \mid t_{n+1} < 1\}$.

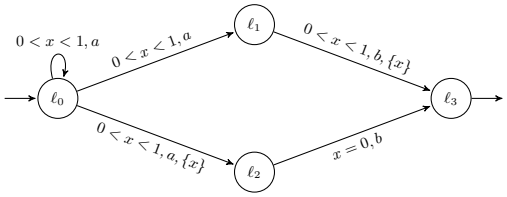

Fig. 1. A timed automaton $\mathcal{A}$

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 $t \in \mathbb{R}_{\geq 0}$ 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 [4]. 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 [2], TAs with integer resets [13] and strongly non-Zeno TAs [3], 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 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 $\mathcal{L}(\mathcal{A})$ [11]. 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 (the *skeleton*) which governs the resetting policy for the clocks of $\mathcal{B}$. The size of obtained deterministic timed automaton $\mathcal{B}$ is also doubly exponential in the size of $\mathcal{A}$.

3 A Game Approach

In [8], 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 $\mathcal{A}$. For simplicity, we present here the method for timed automa without ε-transitions and for which all invariants are true. The general case is presented, for clarity, as extension in Section 4.

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 $2^{L \times \mathsf{Rel}_{M,M'}(X,Y) \times \{\top,\bot\}} \times \mathsf{Reg}^Y_{M'}$ and $2^{L \times \mathsf{Rel}_{M,M'}(X,Y) \times \{\top,\bot\}} \times (\mathsf{Reg}^Y_{M'} \times \Sigma)$;
- $\mathsf{v}_0 = (\{\overline{0}\}, \{(\ell_0, X - Y = 0, \top)\})$ is the initial vertex and belongs to player Spoiler[1];
- $\mathsf{Act} = (\mathsf{Reg}^Y_{M'} \times \Sigma) \cup 2^Y$ 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;
- $\mathsf{Bad} = \{(\{(\ell_j, C_j, \bot)\}_j, r)\} \cup \{(\{(\ell_j, C_j, b_j)\}_j, r) \mid \{\ell_j\}_j \cap F \neq \emptyset \wedge \forall j,\, \ell_j \in F \Rightarrow b_j = \bot\}$ is the set of bad states.

We now detail the edge relation which defines the possible moves of the players. Given $\mathsf{v}_S = (\{(\ell_j, C_j, b_j)\}_j, r) \in \mathsf{V}_S$ 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 $\mathsf{v}_D = (\{(\ell_j, C_j, b_j)\}_j, (r', a)) \in \mathsf{V}_D$ if $\exists (\ell, C, b) \in \{(\ell_j, C_j, b_j)\}_j$ 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

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

$\{(\ell_j, C_j, b_j)\}_j$ 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:

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

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.

As an example, the construction of the game is illustrated on the nondeterministic 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

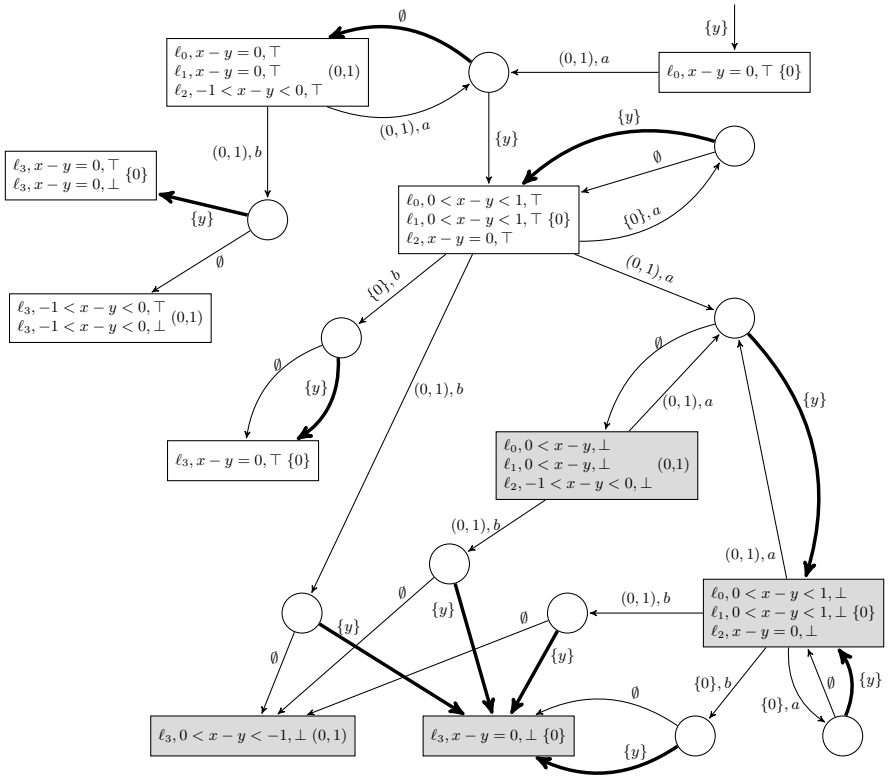

Fig. 2. The game $\mathcal{G}_{\mathcal{A},(1,1)}$ and an example of winning strategy σ for Determinizator

$\bot$ 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 [10]. 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 $\mathcal{A}$ 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 the research report [6]. We however give below its main ideas.

Proof (Sketch of proof). Given a strategy σ, we show that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$ by induction on the length of the runs. The induction step is based on the fact that induced guards for $\mathcal{A}$ and relations always are over-approximated in the game. Moreover, if σ is winning, we show that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathsf{Aut}(\sigma))$. Indeed, any safe state ($\mathsf{v} \notin \mathsf{Bad}$) contains at least one configuration marked $\top$. The latter is necessarily obtained from a configuration marked $\top$ without any over-approximation. Hence any run ending in a safe state has an equivalent run in $\mathcal{A}$.

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.

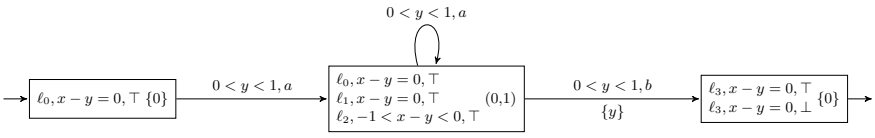

Fig. 3. The deterministic TA $\mathsf{Aut}(\sigma)$ obtained by our construction

Remark 1. Because of the undecidability of the determinizability with fixed resources [14,9], 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 $\mathcal{A}$.

[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.

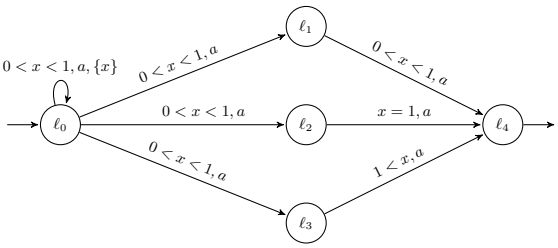

Fig. 4. A determinizable TA for which there is no winning strategy for Determinizator

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 [10]. 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 $x := x'$) [7]. Thus, for every strategy for Determinizator there is 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 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. Let us now briefly explain how to extend the previous construction to deal with ε-transitions and invariants. The extension is presented in details in [6].

ε-transitions. We aim at building an over-approximation without ε-transitions. An ε-closure is performed for each state during the construction of the game. To this attempt, states of the game have to be extended since ε-transitions might be enabled only from some time-successors of the region associated with the state. Therefore, each configuration is associated with a proper region which is a time-successor of the initial region of the state. The ε-closure is effectively computed the same way as successors in the original construction when Determinizator does not reset any clock; computations thus terminate for the same reasons.

Invariants. Ignoring all invariants surely yields an over-approximation. In order to be more precise (while preserving the over-approximation) with each state of the game is associated the most restrictive invariant which contains invariants of all the configurations in the state. In the computation of the successors, invariants are treated similarly to guards and their validity is verified at the transition's target. A state whose invariant is strictly over-approximated is not safe.

5 Comparison with Existing Methods

The method we presented is both more precise than the algorithm of [11] and more general than the procedure of [4]. Let us detail these two points. Note that a deeper comparison with existing work can be found in [6].

5.1 Comparison with [11]

First of all, our method covers the application area of [11] 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 $\mathcal{G}_{\mathcal{A},(k,M')}$.

Moreover, contrary to the method presented in [11], 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 a consequence of the more general fact that, in our approach, a winning strategy can be seen as a timed generalization of the notion of skeleton [11], and solving our game amounts to finding a relevant timed skeleton.

As an example, the algorithm of [11] run on the timed automaton of Figure 1 produces a strict over-approximation, represented on Figure 5.

Our approach also improves the updates of the relations between clocks by taking the original guard into account. Precisely, when computing up_S, an intersection with the guard in the original TA is performed. This improvement allows one, even under the same resetting policy, to refine the over-approximation given by [11].

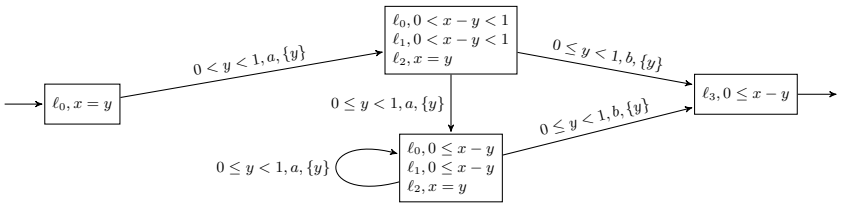

Fig. 5. The result of algorithm [11] on the running example

5.2 Comparison with [4]

Our approach generalizes the one in [4] since, for any timed automaton $\mathcal{A}$ such that the procedure in [4] 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 [4]. Moreover, our approach strictly broadens the class of automata determinized by the procedure of [4] in two respects. First of all, our method allows one to cope with some language inclusions, contrary to [4]. For example, the TA depicted on the left-hand side of Figure 6 cannot be treated by the procedure of [4] 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 [4]. 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 [4], 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 [4].

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

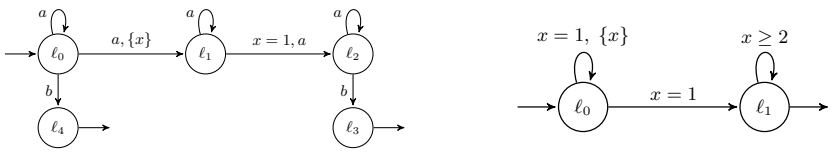

Fig. 6. Examples of determinizable TAs not treatable by [4]

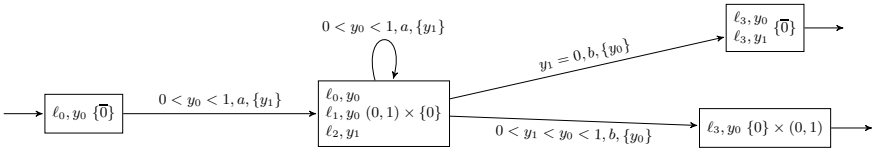

Fig. 7. The result of procedure [4] on the running example

by [4] is depicted in Figure 7: it needs 2 clocks when our method produces a single-clock TA.

The same phenomenon happens with timed automata with integer resets. Timed automata with integer resets, introduced in [13], 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.

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

Intuitively, a single clock is needed to represent clocks of $\mathcal{A}$ since they all share a common fractional part. A complete proof is given in [6].

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 [4] 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 [12].

Last, our method even when restricted to equality relations (conjunctions of constraints of the form $x - y = c$) extends the procedure of [4], whose effective construction when the number of new clocks is fixed, is detailed in the technical report [5]. Note that the latter construction is similar to our approach restricted to mappings instead of relations. We detail in [6] the benefits of (even equality) relations and explain how the sufficient conditions for termination provided in [4] can be weakened in our context.

5.3 Comparison of the Extension with ε-Transition and Invariants

Let us now compare our extended approach with the approach of [11] since the determinization procedure of [4] does not deal with invariants and ε-transitions.

The model in [11] consists of timed automata with silent transitions and actions are classified depending on 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. Note that we classically decided to use invariants to model urgency, but our approach could

be adapted to the same model as the one they use, while preserving urgency much more often, the same way as we do for invariants.

6 Conclusion

In this paper, 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 $\mathcal{A}$. Our construction strictly covers and improves two existing approaches [11,4].

In the research report [6] we 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 for non-deterministic timed automata specifications.

In future work, we plan to investigate how our game-based approach proposed here can be applied to other models and/or other problems.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Alur, R., Fix, L., Henzinger, T.A.: A determinizable class of timed automata. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 1–13. Springer, Heidelberg (1994)
3. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: Proceedings of the 5th IFAC Symposium on System Structure and Control (SSSC 1998), pp. 469–474. Elsevier Science, Amsterdam (1998)
4. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T.: When are timed automata determinizable? In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 43–54. Springer, Heidelberg (2009)
5. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T.: When are timed automata determinizable? Research Report LSV-09-08, Laboratoire Spécification et Vérification, ENS Cachan, France, April 2009, 32 pages (2009)
6. Bertrand, N., Stainer, A., Jéron, T., Krichen, M.: A game approach to determinize timed automata. Research Report 7381, INRIA (September 2010), `http://hal.inria.fr/inria-00524830`
7. Bouyer, P.: From Qualitative to Quantitative Analysis of Timed Systems. Mémoire d'habilitation, Université Paris 7, Paris, France (January 2009)
8. Bouyer, P., Chevalier, F., D'Souza, D.: Fault diagnosis using timed automata. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 219–233. Springer, Heidelberg (2005)
9. Finkel, O.: Undecidable problems about timed automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 187–199. Springer, Heidelberg (2006)

10. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research. LNCS, vol. 2500. Springer, Heidelberg (2002)
11. Krichen, M., Tripakis, S.: Conformance testing for real-time systems. Formal Methods in System Design 34(3), 238–304 (2009)
12. Manasa, L., Krishna, S.N.: Integer reset timed automata: Clock reduction and determinizability. CoRR arXiv:1001.1215v1 (2010)
13. Suman, P.V., Pandya, P.K., Krishna, S.N., Manasa, L.: Timed automata with integer resets: Language inclusion and expressiveness. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 78–92. Springer, Heidelberg (2008)
14. Tripakis, S.: Folk theorems on the determinization and minimization of timed automata. Information Processing Letters 99(6), 222–226 (2006)