A game approach to determinize timed automata

Nathalie Bertrand, Amélie Stainer, Thierry Jéron, Moez Krichen

Form Methods Syst Des
DOI 10.1007/s10703-014-0220-1

A game approach to determinize timed automata

Nathalie Bertrand · Amélie Stainer · Thierry Jéron · Moez Krichen

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 generalizes two recent contributions: the determinization procedure of Baier et al. (Proceedings of the 36th international colloquium on automata, languages and programming (ICALP'09), 2009) and the approximation algorithm of Krichen and Tripakis (Form Methods Syst Des 34(3):238–304, 2009). Moreover, we extend it to apply to timed automata with invariants and ε-transitions, and also consider other useful approximations: under-approximation, and combination of under- and over-approximations.

Keywords Determinization · Timed automata · Approximate determinization · Game

1 Introduction

Computers and software systems are ubiquitous, and it is of great importance that they behave in the expected way. Beyond their logical correctness, most of these systems have to

N. Bertrand · T. Jéron
Inria Rennes Bretagne Atlantique, Rennes, France
e-mail: nathalie.bertrand@inria.fr

T. Jéron
e-mail: thierry.jeron@inria.fr

A. Stainer (✉)
University of Rennes 1, Rennes, France
e-mail: amelie.stainer@gmail.com

M. Krichen
National School of Engineers of Sfax, Sfax, Tunisia
e-mail: moez.krichen@redcad.org

Published online: 16 December 2014

satisfy some timing constraints: they are real-time systems and call for the need of dedicated modelling formalisms. Timed automata (TA), introduced in [1], form a usual model for the specification of real-time embedded systems. Essentially TAs are an extension of finite 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.

A timed automaton is said non-deterministic whenever two runs read the same timed word. The determinization, that is, the construction of an equivalent deterministic timed automaton, is used to address several problems such as implementability, diagnosis or test generation, where the underlying analyses depend on the observable behavior. For example, in the context of offline test generation, the specification has to be determinized in some sense, since the testing artefact needs to foresee the allowed outputs after a trace (i.e. a sequence of observations), thus the set of states after this trace. More generally, restricting to the class of deterministic timed automata makes a lot of problems simpler. In particular, a deterministic timed automaton can easily be complemented, and it is for instance useful for model-checking: given a deterministic timed automaton $\mathcal{A}_\varphi$ representing a property φ, one can easily decide whether another timed automaton satisfies the formula φ by performing the intersection with the complement of $\mathcal{A}_\phi$, and then checking the emptiness of the language of the resulting timed automaton.

However, considering deterministic timed automata only is very restrictive, for several reasons. First of all, non-determinism is often used in an early design process to encompass all possible implementation choices. The synthesis of timed automata, from timed logic may also yield non-deterministic models. Last but not least, the asynchronous composition of deterministic models, e.g. in a distributed system, is typically non-deterministic. This motivates the consideration of non-deterministic specifications.

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 [2,3]. 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 [4], event-clock TAs [5], or TAs with integer resets [6]. In a recent paper, Baier et al. [7] 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 [8]. 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 encode (over-approximate) estimates of possible states of the original TA, and the algorithm uses a fixed policy governed by a finite automaton for resetting clocks.

In this paper we propose a method that combines the approaches of [7] and [8], despite their notable differences. It is inspired by a game-based approach to decide the diagnosability of TAs with fixed resources presented by Bouyer et al. [9]. Similarly to [8], 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 an encoding 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 [7], thus allowing one to enlarge the set of timed automata that can be automatically determinized, thanks to an increased expressive power in the encoding of relations between new and original clocks, and robustness to some language inclusions (e.g. a non-determinizable sub-automaton can be ignored if its language is included in the one for the rest of the timed automaton). Moreover, in contrast to [7], our technique applies to a larger class of timed automata: TAs with ε-transitions and invariants. It is also improves the algorithm of [8] in several respects with an adaptative and timed reset policy, governed by a strategy, compared to a fixed untimed one, and a more precise update of the relations between clocks. The model used in [8] 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 [10]. 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 paper is as follows. In Sect. 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 Sect. 4. Extensions of the method to timed automata with invariants and ε-transitions are then presented in Sect. 5. In Sect. 6, we finally discuss how the construction can be adapted to perform under-approximations, or combinations of under- and over-approximations.

This article is a long version of the paper [11] and extends it in several aspects: we provide the full proofs of our results; the contribution is illustrated on more examples; the comparison to related work is deepened; and beyond over-approximations, we also consider here under-approximations and combinations of over- and under-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 [1].

Given a finite set of clocks X, a clock *valuation* is a mapping $v : X \to \mathbb{R}_+$, where $\mathbb{R}_+$ denotes the set of non-negative reals. 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 $v(x) + t$. For $X' \subseteq X$ we write $v_{[X' \leftarrow 0]}$ for the valuation equal to v on $X \backslash X'$ 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 the 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.

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 $\mathcal{T}_\mathcal{A} = (S, s_0, S_F, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \to)$ where $S = L \times \mathbb{R}_+^X$ is the set of states of the form (ℓ, v), made of a location and a valuation of all clocks in X, $s_0 = (\ell_0, \overline{0})$ is the initial state, $S_F = F \times \mathbb{R}_+^X$ is the set of final states, and $\to \subseteq S \times (\mathbb{R}_+ \cup (\Sigma \cup \{\varepsilon\})) \times S$ is the transition relation composed of the following moves:

– *Discrete moves*: $(\ell, v) \xrightarrow{a} (\ell', v')$ for $a \in \Sigma \cup \{\varepsilon\}$ whenever there exists an edge $(\ell, g, a, X', \ell') \in E$ such that $v \models g \wedge \mathsf{Inv}(\ell)$, $v' = v_{[X' \leftarrow 0]}$ and $v' \models \mathsf{Inv}(\ell')$.
– *Time elapse:* $(\ell, v) \xrightarrow{\tau} (\ell, v + \tau)$ for $\tau \in \mathbb{R}_+$ if $v + \tau \models \mathsf{Inv}(\ell)$.

A *run* ρ of $\mathcal{A}$ is a finite sequence of moves alternating time elapse and discrete moves, starting in s_0 and ending with a discrete move, labeled by an action in Σ, i.e., $\rho = s_0 \xrightarrow{\tau_0} s'_0 \xrightarrow{a_1} s_1 \cdots \xrightarrow{\tau_{k-1}} s'_{k-1} \xrightarrow{a_k} s_k$ with $a_k \neq \varepsilon$. Run ρ is *accepting* if $s_k \in S_F$. A *timed word* over Σ is a sequence $(t_i, b_i)_{i \leq n} \in (\mathbb{R}_+ \times \Sigma)^*$ such that $(t_i)_{i \leq n}$ is non-decreasing. The timed word associated with ρ (defined above) is $w = (t_0, b_0) \ldots (t_m, b_m)$ where $(b_i)_{i \leq m} \in (\mathbb{R}_+ \times \Sigma)^{m+1}$ is the subsequence of labels of the non-ε discrete moves in ρ ($b_i \in \Sigma$) and $t_i = \sum_{j=0}^{i-1} \tau_j$. The τ_j's are the delays elapsed between actions a_j's along ρ, whereas the t_i's are the absolute time at which non-ε actions happen. Conversely, we say that the run ρ *reads* w in $\mathcal{A}$. We write $\mathcal{L}(\mathcal{A})$ for the language of $\mathcal{A}$, that is the set of timed words associated with an accepting run. We say that two timed automata $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* whenever $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. In order to compare behaviors of timed transition systems in a more precise way, we introduce the notion of weak timed simulation relation.

Definition 2 A *weak timed simulation* of a timed transition systems $\mathcal{T}_1 = (S^1, s_0^1, S_F^1, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \to_1)$ by a second timed transition systems $\mathcal{T}_2 = (S^2, s_0^2, S_F^2, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \to_2)$ is a relation $\mathcal{R} \subseteq S^1 \times S^2$ such that:

(1) $(s_0^1, s_0^2) \in \mathcal{R}$,
(2) for all $(s_1, s_2) \in \mathcal{R}$, if $s_1 \in S_F^1$ then $s_2 \in S_F^2$,
(3) for all $(s_1, s_2) \in \mathcal{R}$, for all $b \in \Sigma$, whenever $s_1 \xrightarrow{b}_1 s'_1$, there exists $s'_2 \in S^2$ such that $(s'_1, s'_2) \in \mathcal{R}$ and $s_2 \xrightarrow{b}_2 s'_2$,

(4) for all $(s_1, s_2) \in \mathcal{R}$, for all $\tau_1^1, \cdots \tau_n^1 \in \mathbb{R}_+$, whenever $s_1 \xrightarrow{\tau_1^1}_1 \xrightarrow{\varepsilon}_1 s_1^1 \cdots \xrightarrow{\tau_{n-1}^1}_1 \xrightarrow{\varepsilon}_1 s_{n-1}^1 \xrightarrow{\tau_n^1}_1 s_1'$, there exists $s_2' \in S^2$ and $\tau_1^2, \cdots \tau_m^2 \in \mathbb{R}_+$ such that $(s_1', s_2') \in \mathcal{R}$ and $s' \xrightarrow{\tau_1^2}_2 \xrightarrow{\varepsilon}_2 s_1^2 \cdots \xrightarrow{\tau_{m-1}^2}_2 \xrightarrow{\varepsilon}_2 s_{m-1}^2 \xrightarrow{\tau_m^2}_2 s_2'$ with $\sum_{i=1}^n \tau_i^1 = \sum_{j=1}^m \tau_j^2$.

If such a relation exists, $\mathcal{T}_2$ *weak timed simulates* $\mathcal{T}_1$.

In this paper, we will consider weak timed simulations on timed transition systems induced by timed automata. Given two TAs $\mathcal{A}_1$ and $\mathcal{A}_2$, the existence of a weak timed simulation of $\mathcal{T}_{\mathcal{A}_1}$ by $\mathcal{T}_{\mathcal{A}_2}$, implies the language inclusion $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$.

2.1.1 Determinization of timed automata

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.

Example A running example of a non-deterministic timed automaton is depicted in Fig. 1. This 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: $\mathcal{L}(\mathcal{A}) = \{(t_1, a) \cdots (t_n, a)(t_{n+1}, b) \mid t_{n+1} < 1\}$.

A TA $\mathcal{A}$ 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], and an example of such an automaton is represented in Fig. 2.

Moreover, the determinizability of timed automata is an undecidable problem, even with fixed resources [2,3]. Nevertheless, there are well-known classes of timed automata which are determinizable: event-clock TAs [5] that are timed automata where each clock is associated with one action of the alphabet Σ and thus is reset exactly along transitions over this action; TAs with integer resets [6] that are timed automata where there is a reset along a transition if and only if there is an equality constraint in the guard; and strongly non-Zeno TAs [4] that are timed automata where along every cycle, there is at least one clock which is tested to be larger than a positive constant and reset.

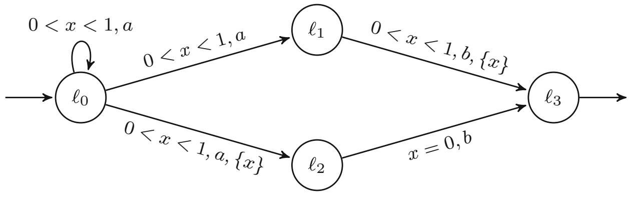


Fig. 1 A timed automaton $\mathcal{A}$

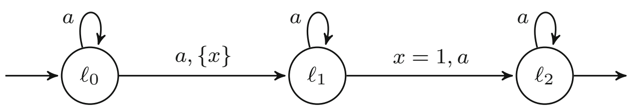


Fig. 2 A non-determinizable timed automaton from [1]

2.1.2 Region abstraction

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 time-successors of r is denoted $\overrightarrow{r}$; formally $\overrightarrow{r} = \{r' \mid \exists v \in r,\ t \in \mathbb{R}_+,\ v' = v + t,\ v' \in r'\}$. Further we define $\overleftrightarrow{r} = \{r' \mid r' \in \overrightarrow{r} \text{ or } r \in \overrightarrow{r'}\}$. A *relation* C over X is the union of the regions intersecting a given conjunction of atomic constraints of the form $x - y \sim c$, where $x, y \in X$, $\sim \in \{<, =, >\}$ and $c \in \mathbb{N}$. Assuming all constants c belong to $[-M, M]$, for some integer $M \in \mathbb{N}$ we write $\mathsf{Rel}_M(X)$ for the set of relations over X. Relations will be used in the sequel to express constraints between the clocks of two timed automata, hence the name.

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

2.1.2.1 Diagonal constraints In the literature, guards in timed automata are sometimes more general than the ones we introduced in $G_M(X)$ and can contain atomic constraints of the form $x - y \sim c$ for $x, y \in X$, $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in [0, M] \cap \mathbb{N}$. These so called *diagonal constraints* do not extend the expressive power of timed automata: from any timed automaton with diagonal constraints, one can build an equivalent one without diagonal constraints (potentially with an exponential blowup of the state space). In the sequel, we almost always consider timed automata without diagonal constraints, with the exception of the timed automata our game approach constructs.

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.

2.2.1 Determinization procedure

An abstract determinization procedure which effectively constructs a deterministic timed automaton for several classes of determinizable timed automata is presented in [7]. 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. Assuming that at each level of the tree, only a finite number of clocks is used (this condition is the 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 by the determinizable syntactic classes of timed automata presented in Sect. 2.1 (event-clock, integer reset and strongly non-Zeno), which can thus be determinized by this procedure. In general, the resulting deterministic timed automaton is doubly exponential in the size of $\mathcal{A}$: more precisely, the number of locations of the resulting automaton is doubly exponential in its number of clocks and in the number of clocks of $\mathcal{A}$ and exponential in the number of locations of $\mathcal{A}$.

2.2.2 Deterministic over-approximation

By contrast, Krichen and Tripakis propose in [8] 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 $\mathcal{L}(\mathcal{A})$. Given a set of new clocks, the TA $\mathcal{B}$ is built by simulation of $\mathcal{A}$ using only information stored in the new clocks. A location of $\mathcal{B}$ constitutes a state estimate of $\mathcal{A}$ consisting of a (generally infinite but finitely represented) 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 the skeleton. It governs the resetting policy for the clocks of $\mathcal{B}$, so that the resets only depend on the untimed word. Here also the TA $\mathcal{B}$ is doubly exponential in the size of $\mathcal{A}$.

3 A game approach to determinization

Intuitively, the subset construction, which successfully determinizes finite atomata, fails for timed automata because of non-uniform resets. When performing a subset construction, it could thus be necessary to use an unbounded number of clocks to store information from all possible paths so far. This phenomenon on a particular timed automaton indicates that it is not determinizable by the approach of [7]. Our objective is to design a finer approach, yet the main problem remains to find a sufficient number of clocks and suitable resets to preserve all the timing information needed for the subset construction. One key feature of our approach lies in the use of relations between the clocks of the input timed automaton and the ones of the deterministic over-approximation, to encode the important timing information.

Our approach is partly inspired by [9] in which, given a plant—modeled by a timed automaton—and fixed resources, the authors build a game where one 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 $\mathcal{A}$, over some resources, and given fixed resources (k, M'), we derive a game between two players Spoiler and Determinizator, such that if Determinizator has a winning strategy, then a deterministic timed automaton $\mathcal{B}$, over resources (k, M'), 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 clarity, we first expose the method for timed automa without ε-transitions and in which all invariants are true. The general case is presented as an extension in Sect. 5.

3.1 Game definition

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. Note that the definition can be difficult to read, but some details of the construction of the game for the small timed automaton in Fig. 1 with a single clock are then given to illustrate the different steps.

Let Y be a set of clocks, disjoint from X, and of cardinality k, which will serve as set of clocks of $\mathcal{B}$. The encoding of clocks of $\mathcal{A}$ by clocks of $\mathcal{B}$ will be described through relations over $X \cup Y$.

The idea is to perform a subtle subset construction using relations to try to determinize $\mathcal{A}$. Using unions of regions (with a fixed maximal constant) instead of conjunctions of diagonal constraints allows to deal with a finite number of relations, in the same way as for regions. In the sequel, we often abuse notations and write conjunctions of constraints rather than unions of regions, for readability. As an example, $\bigwedge_{z,z' \in Z} z - z' = 0$ represents the union of all regions in which all clocks are equal, together with the unbounded region $(M, \infty)^Z$. In our game construction, relations are updated using the operation $\overleftrightarrow{R}$ which assigns to a union of regions R, the union of all time-successors and time-predecessors of regions in R.

States of the game (future locations of the resulting deterministic timed automaton) are state estimates, symbolically represented using locations of $\mathcal{A}$ and regions over clocks in Y together with relations for the clocks in $X \cup Y$. The risk of over-approximation is marked and propagated thanks to booleans.

The initial state of the game is a state of Spoiler consisting of a single configuration with location ℓ_0 (initial location of $\mathcal{A}$), the simplest relation over $X \cup Y$: $\forall z, z' \in X \cup Y, z - z' = 0$, and the 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 $a \in \Sigma$, representing the fact that Spoiler chooses to read an a in the region r. Determinizator answers by deciding the set of clocks $Y' \subseteq Y$ it 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 on clocks in $X \cup Y$, and a boolean marking ($\top$ or $\bot$). A state of Spoiler thus constitutes a state 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 state estimate of Spoiler together with the move of Spoiler.

Bad states player Determinizator wants to avoid are those where an over-approximation may have occurred, namely, 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 $\bot$.

Definition 3 Let $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E)$ be a timed automaton and (k, M') some given resources. We let $\mathbf{M} = \max(M, M')$, and Y be a set of k clocks. The game associated with $\mathcal{A}$ and (k, M') is $\mathcal{G}_{\mathcal{A},(k,M')} = (\mathsf{V}, \mathsf{v}_0, \mathsf{Act}, \delta, \mathsf{Bad})$ where:

- $\mathsf{V} = \mathsf{V}_S \sqcup \mathsf{V}_D$ is a finite set of vertices, made of the disjoint union of the set $\mathsf{V}_S \subseteq 2^{L \times \mathsf{Rel}_{\mathbf{M}}(X \cup Y) \times \{\top, \bot\}} \times \mathsf{Reg}^Y_{M'}$ of vertices of Spoiler, and the set $\mathsf{V}_D \subseteq \mathsf{V}_S \times \mathsf{Reg}^Y_{M'} \times \Sigma$ of vertices of Determinizator.
- $\mathsf{v}_0 = (\{(\ell_0, \bigwedge_{z,z' \in X \cup Y} z - z' = 0, \top)\}, \{\overline{0}\})$ is the initial vertex; $\mathsf{v}_0 \in \mathsf{V}_S$;
- Act is the set of possible actions partitioned into $\mathsf{Act}_S = \mathsf{Reg}^Y_{M'} \times \Sigma$ and $\mathsf{Act}_D = 2^Y$;
- $\delta = \delta_S \cup \delta_D$ is the transition relation with δ_S and δ_D defined as follows:

- $\delta_S \subseteq \mathsf{V}_S \times \mathsf{Act}_S \times \mathsf{V}_D$ consists of all edges of the form $(\mathcal{E}, r) \xrightarrow{(r',a)} ((\mathcal{E}, r), (r', a))$ if $r' \in \overrightarrow{r}$ and there exist $(\ell, C, b) \in \mathcal{E}$ and $\ell \xrightarrow{g,a,X'} \ell' \in E$ such that
$$[r' \cap C]_{|X} \cap g \neq \emptyset \ ; \qquad (\mathsf{C}_{\neq\emptyset})$$
- $\delta_D \subseteq \mathsf{V}_D \times \mathsf{Act}_D \times \mathsf{V}_S$ is the set of edges of the form $((\mathcal{E}, r), (r', a)) \xrightarrow{Y'} (\mathcal{E}', r'_{[Y' \leftarrow 0]})$ where $\mathcal{E}' = \cup_{\gamma \in \mathcal{E}} \mathsf{Succ}_e[r', a, Y'](\gamma)$ and Succ_e is the elementary successor function:
$$\mathsf{Succ}_e[r', a, Y'](\ell, C, b) = \left\{ (\ell', C', b') \mid \exists \ell \xrightarrow{g,a,X'} \ell' \in E \text{ s.t. } \begin{bmatrix} [r' \cap C]_{|X} \cap g \neq \emptyset \\ C' = \mathsf{up}(r', C, g, X', Y') \\ b' = b \wedge ([r' \cap C]_{|X} \subseteq g) \end{bmatrix} \right\}, \quad (1)$$
with up the update function for relations
$$\mathsf{up}(r', C, g, X', Y') = \overleftrightarrow{(r' \cap C \cap g)_{[X' \leftarrow 0][Y' \leftarrow 0]}} \ ; \qquad (2)$$
- $\mathsf{Bad} \subseteq \mathsf{V}_S$ is the set of bad vertices, defined by $\mathsf{Bad} = \big\{(\{(\ell_j, C_j, \bot)\}_j, r)\big\} \cup \big\{(\{(\ell_j, C_j, b_j)\}_j, r) \mid \exists j,\ \ell_j \in F \wedge \forall j,\ \ell_j \in F \Rightarrow b_j = \bot\big\}$.

The objective of the game for player Determinizator is to avoid the set Bad. Player Spoiler has the opposite objective.

Let us comment the definition of the game. The edge relation δ gives the possible moves for each player and is deterministic: for every $\mathsf{v}_S \in \mathsf{V}_S$ and $(r', a) \in \mathsf{Act}_S$ there is a single successor vertex $\mathsf{v}_D \in \mathsf{V}_D$ such that $(\mathsf{v}_S, r', a, \mathsf{v}_D) \in \delta$ and for every $\mathsf{v}_D \in \mathsf{V}_D$ and $Y' \in \mathsf{Act}_D$ there is a single successor vertex $\mathsf{v}_S \in \mathsf{V}_S$ such that $(\mathsf{v}_D, Y', \mathsf{v}_S) \in \delta$. We now detail how these successors are defined.

A state $\mathsf{v}_S = (\mathcal{E}, r) \in \mathsf{V}_S$ is composed of a state estimate $\mathcal{E}$ together with a region over Y, and elements of $\mathcal{E}$ are called *configurations*. Given $\mathsf{v}_S = (\mathcal{E}, r) \in \mathsf{V}_S$ a state of Spoiler and $(r', a) \in \mathsf{Act}_S$ 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 the successors of this state have a non-empty set of configurations (condition $\mathsf{C}_{\neq\emptyset}$).

Given $\mathsf{v}_D = (\mathcal{E}, (r', a)) \in \mathsf{V}_D$ a state of Determinizator and $Y' \in \mathsf{Act}_D$ 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 $(\ell, C, b) \in \mathcal{E}$ by (r', a) and after resetting Y': $\mathcal{E}' = \big\{\mathsf{Succ}_e[r', a, Y'](\ell, C, b) \mid (\ell, C, b) \in \mathcal{E}\big\}$. The formal definition of the elementary successor function is given above in Eq. (1) for a configuration (ℓ, C, b).

$\mathsf{up}(r', C, g, X', Y')$ is the update of the relation on clocks in $X \cup Y$ after the moves of the two players, that is after taking action a in r', resetting $X' \subseteq X$ and $Y' \subseteq Y$, and ensuring the satisfaction of g. The resulting updated relation is also formally defined above, in Eq. (2), as a union of regions on $X \cup Y$ with maximal constant $\mathbf{M}$. In the update, the intersection with g aims at stopping runs that for sure will correspond to timed words outside of $\mathcal{L}(\mathcal{A})$. Taking the time-successor and time-predecessor permits to obtain a relation. 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.

The boolean b keeps track of potential over-approximations. Boolean b' is set to $\bot$ if either $b = \bot$ or the induced guard $[r' \cap C]_{|X}$ over-approximates g: $\neg([r' \cap C]_{|X} \subseteq g)$ (this condition is written $\mathsf{C}_{\neg\subseteq}$ for short). Therefore, if the boolean is $\top$, necessarily there has been no over-approximations. In a state, it is thus sufficient to have a configuration with boolean $\top$ to know that the language was not over-approximated. The set Bad is defined accordingly, and avoiding it ensures that at each step, but also for the acceptance condition, the language is not overapproximated.

3.1.1 Size of $\mathcal{G}_{\mathcal{A},(k,M')}$

State estimates are sets of configurations, each of which contains a relation over $X \cup Y$. Therefore, the number of states in $\mathcal{G}_{\mathcal{A},(k,M')}$ is doubly exponential in the size of $\mathcal{A}$. Also, Determinizator's states have exponentially many outgoing edges in k, the size of Y. We will see in Proposition 2 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. Nevertheless, the complexity order is not impacted and the size of the resulting deterministic timed automaton could even be larger than with multiple resets.

3.2 Example

As an example, the construction of the game is illustrated on the non-deterministic timed automaton $\mathcal{A}$ depicted in Fig. 1, page 5. Part of the construction of the associated game $\mathcal{G}_{\mathcal{A},(1,1)}$ is represented in Fig. 3. $\mathcal{A}$ has a single clock called x, and the game uses a single clock y (for simplicity, but the construction would work with an arbitrary number of clocks). Rectangular states belong to Spoiler and circular ones to Determinizator. Note that, to simplify the picture, the labels of states of Determinizator are omitted (recall that they contain the predecessor state together with the move of Spoiler).

Let us detail the first steps of the game construction. The initial state v_0 contains the single configuration $(\ell_0, x-y = 0, \top)$ together with the initial region $\{0\}$ over $\{y\}$. The computation of the possible moves of Spoiler from the initial state is performed by examination of the outgoing transitions from ℓ_0 in $\mathcal{A}$: they all read letter a and are guarded by $0 < x < 1$. Given the relation $x - y = 0$, this guard can be expressed using clock y without any approximation: $0 < y < 1$. Thus, the only possible move for Spoiler from the initial state of $\mathcal{G}_{\mathcal{A},(1,1)}$ is $(0 < y < 1, a)$ and the successor configurations will still have $\top$ as boolean, reflecting that no overapproximation happened so far (condition $\mathsf{C}_{\neg\subseteq}$ is not fulfilled). The successor state by this move is a state of Determinizator, defined as the pair formed of its predecessor state v_0 and the move of Spoiler $((0, 1), a)$.

From there, Determinizator has two possible moves: resetting y or not. We explain the computation of the successor configurations by the move $\emptyset$ of Determinizator, yielding the state v_1. Since in $\mathcal{A}$, from location ℓ_0, there are three transitions corresponding to the move $(0 < y < 1, q)$ of Spoiler, three successor configurations need to be computed. The transitions to locations ℓ_0 and ℓ_1 do not reset x, and Determinizator chooses not to reset y, thus the relation for the corresponding configurations remains $x - y = 0$. For the last configuration, associated with the target location ℓ_2, x is reset in $\mathcal{A}$, but y is not and $0 < y < 1$, so the derived relation is $-1 < x - y < 0$. Recall that the markers of all the configurations are $\top$ because the guard $0 < x < 1$ has not been approximated. Last, the region on $\{y\}$ of v_1 is naturally $0 < y < 1$.

In the state v_2 obtained when y is reset by Determinizator, the relations differ. For the configuration with location ℓ_2 the relation is simply $x - y = 0$ since both x and y were reset. The two other configurations share the relation $0 < x - y < 1$ derived from $0 < x < 1$ and y reset. In v_2 again all the booleans are true and the associated region is $\{0\}$. Note that

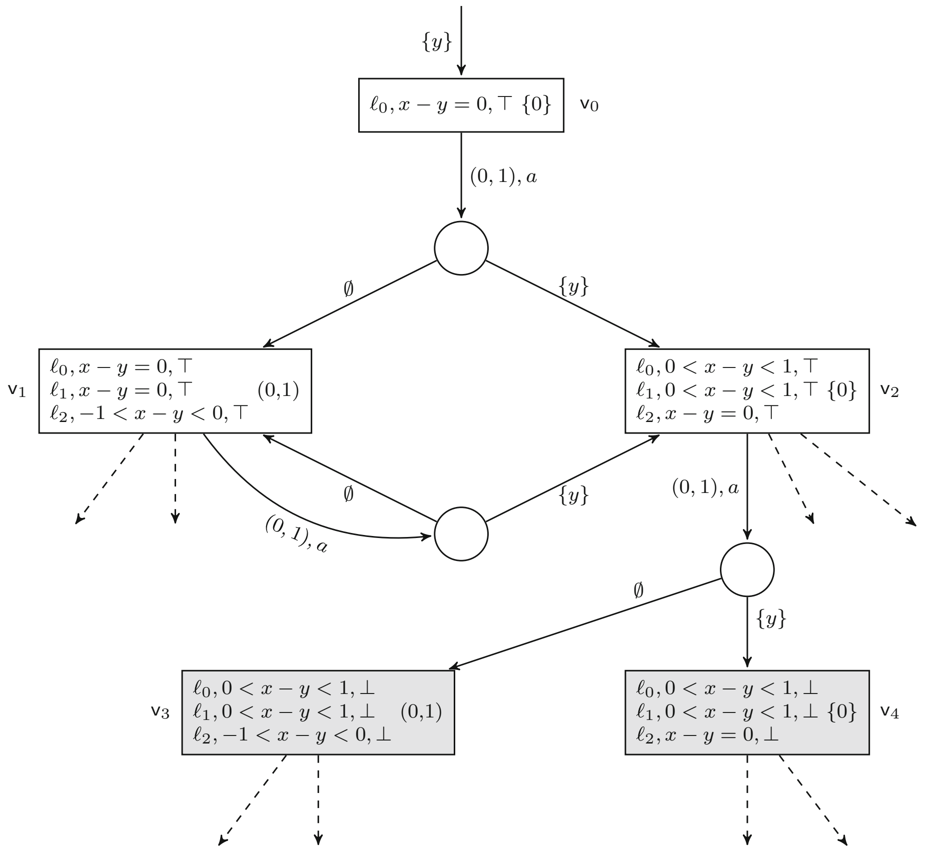


Fig. 3 Excerpt the game $\mathcal{G}_{\mathcal{A},(1,1)}$

from v_1, the move $(0 < y < 1, a)$ of Spoiler yields exactly the same successors as from v_0. Indeed, the only relevant configuration when firing action a is the one with location ℓ_0 (because there no a-transition can be fired from ℓ_1 or ℓ_2) and the configuration associated with ℓ_0 in v_1 is exactly the same as the unique configuration in v_0.

We end this example by detailing the construction of the successors for v_2, assuming Spoiler chose the move $(0 < y < 1, a)$. Here also the only relevant configuration in v_2 is $(\ell_0, 0 < x - y < 1, \top)$, because there are no a transitions in $\mathcal{A}$ from ℓ_1 and ℓ_2. Since the relation $0 < x - y < 1$ is different from $x - y = 0$, the guard on x induced by the region $0 < y < 1$ is not trivial. Figure 4 illustrates the computation of the guard over x induced by the relation $C = 0 < x - y < 1$ and the region $r' = 0 < y < 1$. The dotted area represents the set of valuations over $\{x, y\}$ which satisfy $0 < y < 1$, and the dashed area represents the relation $C = 0 < x - y < 1$. The induced guard $[r' \cap C]_{|\{x\}}$ (i.e. the guard over x encoded by the guard r' on y through the relation C) is then the projection over clock x of the intersection of these two areas. In this example, the induced guard is $0 < x < 2$. Note that the figure represents the computation of the induced guard, *without* taking into account the maximal constant. Since the maximal constant in this example is 1, the real induced guard will be $0 < x$ rather than $0 < x < 2$. Therefore, the transitions of $\mathcal{A}$ corresponding to the choice of Spoiler $(0 < y < 1, a)$ are the three possible transitions with source ℓ_0, but they

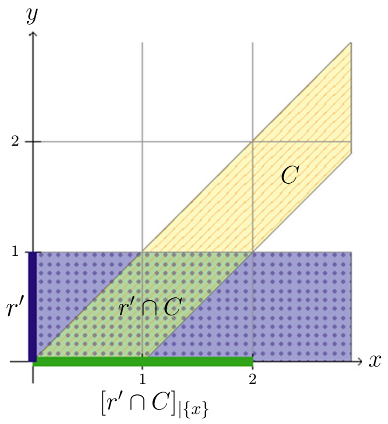


Fig. 4 Construction of the induced guard

are over-approximated. Indeed, the induced guard $[r' \cap C]_{|\{x\}} = 0 < x < 2$ is not included in the original guard $g = 0 < x < 1$ in $\mathcal{A}$, i.e., the region r' possibly encodes more values than the guard g. As a consequence, all the configurations in v_3 and v_4 are marked $\perp$, and thus these states belong to Bad, represented by the grey color in Fig. 3.

It remains to detail the computation of the relations in v_3 and v_4. Assuming Determinizator chooses not to reset y leads to v_3, in which for the configuration with location ℓ_0, the relation is the smallest one containing $(0 < x - y < 1) \cap (0 < y < 1) \cap (0 < x < 1)$, that is $0 < x - y < 1$. The relation for the last configuration is $\overleftrightarrow{\big((0 < x - y < 1) \cap (0 < y < 1) \cap (0 < x < 1)\big)_{[x \leftarrow 0]}}$, which is same as $\overleftrightarrow{(x = 0 < y < 1)}$, that is $-1 < x - y < 0$.

The complete construction of the game $\mathcal{G}_{\mathcal{A},(1,1)}$ is depicted in Fig. 5, together with a winning strategy for Determinizator represented by bold edges.

3.3 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 for Determinizator. The possible behaviors in the game are expressed by means of strategies. Intuitively, a strategy for player Determinizator (resp. Spoiler) chooses which move to perform from vertex $\mathsf{v}_D \in \mathsf{V}_D$ (resp. $\mathsf{v}_S \in \mathsf{V}_S$) based on the history of the game so far. It is a classical result that, for safety games, winning strategies can be chosen positional (the chosen move only depends on the current vertex) and they can be computed in linear time in the size of the arena [12]. Therefore, in the following, we only consider positional strategies, and simply write "strategies" for "positional strategies".

A strategy for player Determinizator is thus described by a function $\sigma : \mathsf{V}_D \to \mathsf{Act}_D$ assigning to each state $\mathsf{v}_D \in \mathsf{V}_D$ a set $Y' \subseteq Y$ of clocks to reset; the successor state is then $\mathsf{v}_S \in \mathsf{V}_S$ such that $(\mathsf{v}_D, Y', \mathsf{v}_S) \in \delta$. Symmetrically, a strategy for Spoiler is a mapping $\sigma' : \mathsf{V}_S \to \mathsf{Act}_S$ assigning to each state $\mathsf{v}_S \in \mathsf{V}_S$ a region over Y and an action $a \in \Sigma$; the successor state is then $\mathsf{v}_D \in \mathsf{V}_D$ such that $(\mathsf{v}_S, (r', a), \mathsf{v}_D) \in \delta$. A pair of strategies (σ, σ') (one for each of the players) yields a path, written $\pi_{\sigma,\sigma'}$ in the game graph, which is finite or has a lasso shape: a prefix path starting from v_0 followed by a cycle. A strategy σ for Determinizator is *winning* if whatever the strategy σ' for Spoiler, the path $\pi_{\sigma,\sigma'}$ does not visit any Bad states.

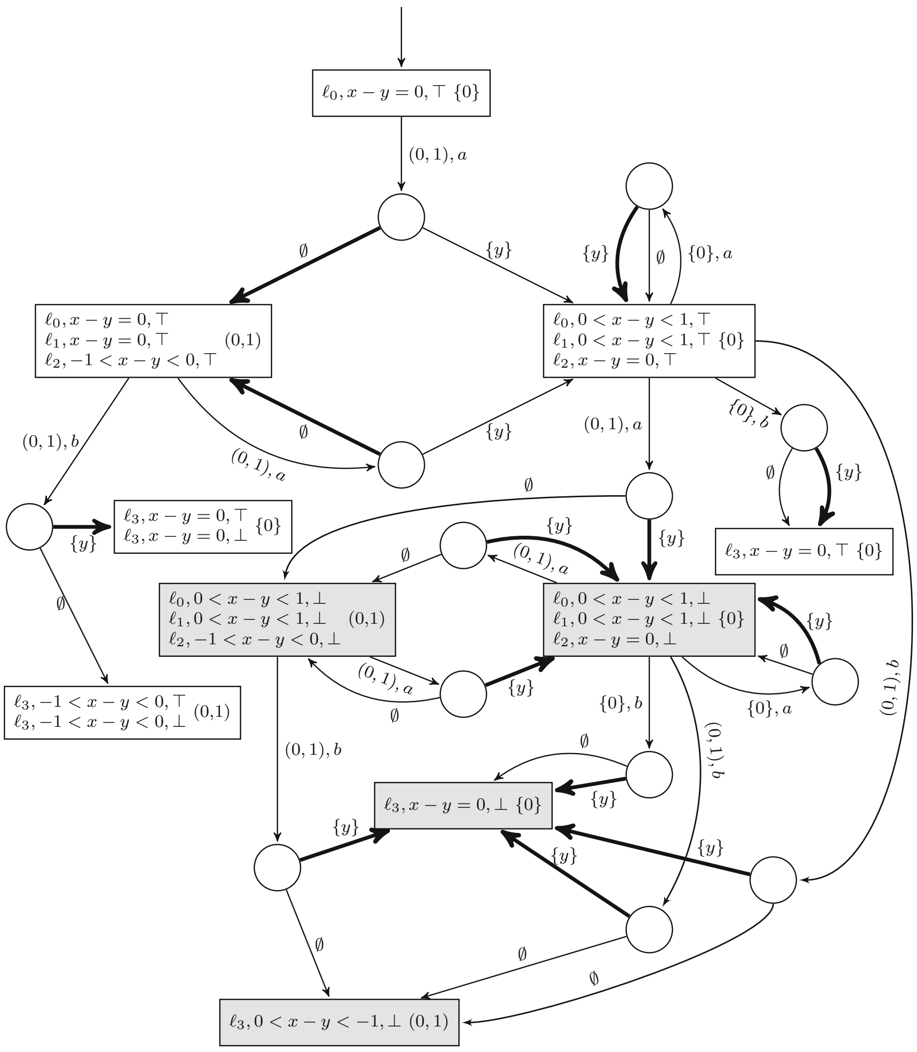


Fig. 5 The game $\mathcal{G}_{\mathcal{A},(1,1)}$ and an example of winning strategy σ for Determinizator

With every strategy for Determinizator σ is associated the timed automaton $\mathsf{Aut}(\sigma)$ obtained by merging a transition of Spoiler with the transition chosen by Determinizator just after, and setting as final locations those states of Spoiler containing at least one final location of $\mathcal{A}$.

Definition 4 Let $\mathcal{G}_{\mathcal{A},(k,M')} = (\mathsf{V}, \mathsf{v}_0, \mathsf{Act}, \delta, \mathsf{Bad})$ be the game built from a timed automaton $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E)$ and resources (k, M'). With a strategy for Determinizator $\sigma : \mathsf{V}_D \to \mathsf{Act}_D$, is associated the timed automaton $\mathsf{Aut}(\sigma) = (\mathsf{V}_S, \mathsf{v}_0, F', \Sigma, Y, M', E')$ defined by:

- V_S is the set of locations, with v_0 the initial location;
- $F' = \{\mathsf{v}_S = (\mathcal{E}, r) \in \mathsf{V}_S \mid \exists(\ell, C, b) \in \mathcal{E} : \ \ell \in F\}$ is the set of final locations;
- Y is the set of k clocks used in $\mathcal{G}_{\mathcal{A},(k,M')}$;

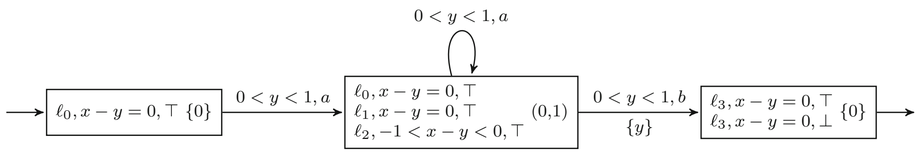

$0 < y < 1, a$
$\ell_0, x - y = 0, \top$ $\{0\}$
$0 < y < 1, a$
$\ell_0, x - y = 0, \top$
$\ell_1, x - y = 0, \top$
$\ell_2, -1 < x - y < 0, \top$
(0,1)
$0 < y < 1, b$
$\{y\}$
$\ell_3, x - y = 0, \top$
$\ell_3, x - y = 0, \bot$
$\{0\}$

Fig. 6 The deterministic TA $\mathsf{Aut}(\sigma_{\mathsf{Win}})$ obtained by our construction

– the set of edges is $E' = \{(\mathsf{v}_S, g, a, Y', \mathsf{v}'_S) \in \mathsf{V}_S \times G_{M'}(Y) \times \Sigma \times 2^Y \times \mathsf{V}_S \mid \exists \mathsf{v}_D \in \mathsf{V}_D$ such that $(\mathsf{v}_S, (g, a), \mathsf{v}_D) \in \delta,\ (\mathsf{v}_D, Y', \mathsf{v}'_S) \in \delta$ and $\sigma(\mathsf{v}_D) = Y'\}$.

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 with no ε-transition and no invariant, and (k, M') resources. 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 Sect. 5.3; however the main ideas for this simpler case are given below.

Proof (Sketch) Given a strategy σ for Determinizator, we show that there exists a weak timed simulation between $\mathcal{A}$ and $\mathsf{Aut}(\sigma)$, namely the relation ρ defined by: $\mathcal{R} = \{((\ell, v), ((\mathcal{E}, r), v')) \mid \exists (\ell, C, b) \in \mathcal{E},\ (v, v') \in C \wedge v' \in \overrightarrow{r}\}$. This entails the language inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$.

Assuming now that σ is winning, given a run ρ in $\mathsf{Aut}(\sigma)$, one can build backwards a path in $\mathcal{A}$, from an accepting configuration to the initial one by application of elementary predecessors. Since σ is winning, guards are not over-approximated in $\mathsf{Aut}(\sigma)$, and there is a run in $\mathcal{A}$ with the same delays as in ρ and following the path. This entails the reverse language inclusion $\mathcal{L}(\mathsf{Aut}(\sigma)) \subseteq \mathcal{L}(\mathcal{A})$. □

Standard techniques based on the computation of attractors allow one to check for the existence of a winning strategy for Determinizator, and in the positive case, to extract such a strategy [12]. Our game construction can thus be applied to construct deterministic equivalent (or deterministic over-approximations) to timed automata. On our running example, on Fig. 5, a winning strategy σ_{Win} for Determinizator is represented by the bold edges. This strategy yields the deterministic equivalent for $\mathsf{Aut}(\sigma_{\mathsf{Win}})$ depicted in Fig. 6.

In the approach for the diagnosability problem [9] from which our game construction is inspired, the existence of a winning strategy is equivalent to the existence of a diagnoser with given resources. In comparison, since the determinizability under fixed resources is undecidable, we cannot hope for a reciprocal statement to the one of Theorem 1.

Proposition 1 *There exists a TA $\mathcal{A}$ that admits an equivalent deterministic TA over resources (k, M'), and such that Determinizator has no winning strategy in $\mathcal{G}_{\mathcal{A},(k,M')}$.*

To illustrate this phenomenon, Fig. 7 represents a timed automaton $\mathcal{A}$ which is determinizable with resources (1, 1), but for which Determinizator has no winning strategy in $\mathcal{G}_{\mathcal{A},(1,1)}$. Intuitively the self loop on ℓ_0 forces Determinizator to reset the clock in its first move; afterwards, on each branch of the automaton (via ℓ_1, ℓ_2 or ℓ_3), the behavior of $\mathcal{A}$ is strictly over-approximated in the game. However, each over-approximation on a branch is

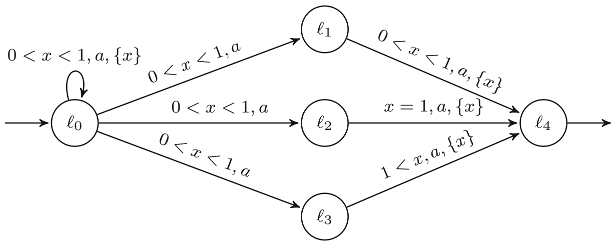


Fig. 7 A determinizable TA for which there is no winning strategy for Determinizator

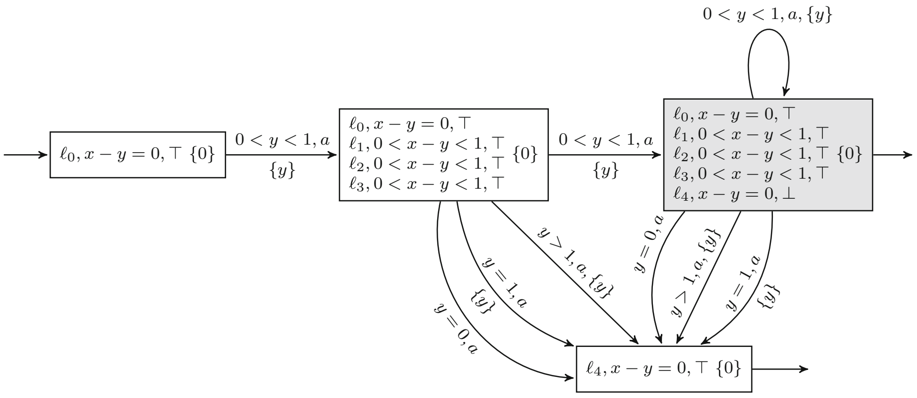

$\ell_0, x - y = 0, \top$ $\{0\}$
$0 < y < 1, a$
$\{y\}$
$\ell_0, x - y = 0, \top$
$\ell_1, 0 < x - y < 1, \top$
$\ell_2, 0 < x - y < 1, \top$ $\{0\}$
$\ell_3, 0 < x - y < 1, \top$
$0 < y < 1, a$
$\{y\}$
$0 < y < 1, a, \{y\}$
$\ell_0, x - y = 0, \top$
$\ell_1, 0 < x - y < 1, \top$
$\ell_2, 0 < x - y < 1, \top$ $\{0\}$
$\ell_3, 0 < x - y < 1, \top$
$\ell_4, x - y = 0, \bot$
$y = 0, a$
$y = 1, a$
$\{y\}$
$y > 1, a, \{y\}$
$y = 0, a$
$y > 1, a, \{y\}$
$y = 1, a$
$\{y\}$
$\ell_4, x - y = 0, \top$ $\{0\}$

Fig. 8 A deterministic equivalent to the TA in Fig. 7 obtained with a losing strategy

"covered" by the other branches, so that the losing strategy yields a deterministic equivalent to $\mathcal{A}$, represented on Fig. 8. However, if one sets the resources to (2, 1), Determinizator has a winning strategy in $\mathcal{G}_{\mathcal{A},(2,1)}$. This example suggests an iterative approach to construct a deterministic equivalent. One starts with resources (1, 1) and increases the resources until a winning strategy is found. However, we have no guarantee that this method will indeed terminate. It remains an open problem whether given a determinizable timed automaton, there exist resources so that the resulting game admits a winning strategy.

Remark 1 The size of the game is doubly exponential in the size of the original timed automaton and we do not have a better upper bound for the resulting deterministic timed automaton. Note that any deterministic timed automaton $\mathsf{Aut}(\sigma)$ has diagonal guards, since its transitions are guarded by regions over Y. Yet, this diagonal guards can be removed avoiding the traditional exponential blowup, because the satisfaction of diagonal constraints is already encoded in the region associated with each location. From a timed automaton without diagonal guards, our method thus allows one to construct a deterministic over-approximation without diagonal guards.

3.3.1 Atomic resets

We now establish that winning strategies for Determinizator can be chosen in the restricted class of positional strategies with atomic resets. A strategy σ for Determinizator has atomic resets if for every move $Y' \subseteq Y$ in σ, $|Y'| \leq 1$: in words, at most one clock is reset on each move of Determinizator.

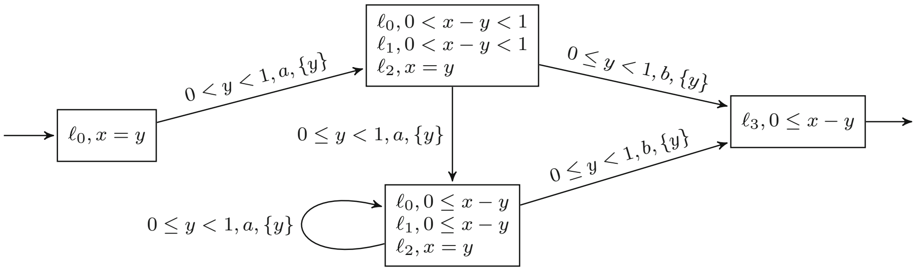


Fig. 9 The result of algorithm [8] on the running example

Proposition 2 *Determinizator has a winning strategy* $\sigma : \mathsf{V}_D \to \mathsf{Act}_D$ *if and only if it has a winning strategy with atomic resets* $\sigma' : \mathsf{V}_D \to Y \cup \{\emptyset\}$.

Proof (sketch) The proof only treats the direct implication because the other one is trivial. Intuitively two clocks of Y with the same value do not give more information than a single one, so that it is never worth resetting two or more clocks. More precisely, any timed automaton can be turned into a weakly timed bisimilar one with atomic resets only, using a construction similar to the one that removes clock transfers (i.e., updates of the form $x := x'$) [13]. $\square$

We gave the definition of our game approach for the determinization of timed automata without invariants and ε-transitions. Before extending the procedure to a richer model, we compare, in Sect. 4 this first version to the two existing approaches introduced for this class of timed automata.

4 Comparison with existing methods

In this section, we compare the game approach presented in Sect. 3 to the existing methods: on the one hand, the approximate determinization algorithm from [8] and on the other hand the determinization procedure from [7]. We argue that our approach often improves the algorithm of [8] and is more general than the procedure of [7].

4.1 Comparison with [8]

In [8] the construction of a deterministic over-approximation is guided by a *skeleton*, a finite automaton which governs the clock resets in the deterministic timed automaton in construction. The resets are thus defined by a regular untimed language. Strategies can be seen as skeletons with additional timing information, since the resets also depend on the regions the actions are taken in. Moreover the game allows us to choose a good strategy, contrary to the a priori fixed skeletons. On the running example, no skeleton would imply an exact determinization by [8]. As an example, we depict, on Fig. 9, the resulting strict over-approximation, when a single clock y is used and reset after each action. In contrast, our approach, with the resources (1, 1), is exact.

Our approach improves the precision of the relations between clocks by taking the original guard into account when computing the updated relation. Precisely, an intersection with the guard in the original TA is performed during the computation of the update up. This refinement of the update computation could be incorporated into [8]'s algorithm.

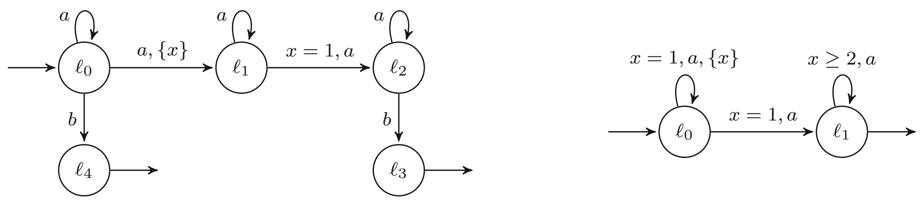


Fig. 10 Examples of determinizable TA not treatable by [7]

However, this does not guarantee that our methodology returns tighter over-approximations. It remains an open problem to compare the outputs of the two methods in case there is no winning strategy. Recall that if a game admits no winning strategy, we cannot conclude that no deterministic equivalent over the given resources exist. Unfolding the game (i.e. using additional finite memory) cannot yield the existence of a winning strategy, yet it may imply the existence of a strategy yielding a tighter approximation or even an exact determinization. Therefore, it could be that the algorithm by [8] produces an exact determinization while our approach doesn't, whatever the strategy. Indeed, unfolding the behaviour of the input timed automaton, skeletons may induce an unfolding of the timed automaton, which we cannot do.

Combining the finite-state skeleton of [8] and our game-approach by incorporating a finite memory in the game would be a way to really subsume the previously existing over-approximation method.

Also, when a winning strategy exists, we know that the determinization is exact, while there is no such criterion in the work of Krichen and Tripakis [8].

4.2 Comparison with [7]

Our approach generalizes the one in [7] since, for any timed automaton $\mathcal{A}$ such that the procedure in [7] yields an equivalent deterministic timed automaton with k clocks and maximal constant M', there is a winning strategy for Determinizator in $\mathcal{G}_{\mathcal{A},(k,M')}$. Intuitively this is a consequence of the fact that relations between clocks of $\mathcal{A}$ and clocks in the game generalize the mappings from [7], since a mapping can be seen as a restricted relation, namely a conjunction of constraints of the form $x - y = 0$.

Moreover, our approach strictly broadens the class of automata determinized by the procedure of [7] in two respects.

- First of all, our method allows one to cope with some language inclusions. For example, the TA depicted on the left-hand side of Fig. 10 cannot be treated by the procedure of [7] 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 [7]. However, all timed words accepted in ℓ_3 are also 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 thanks to the boolean ($\top$ or $\bot$) associated with each configuration, and will thus provide an equivalent deterministic timed automaton. This determinized version of the TA from Fig. 10, left, was computed using a prototype implementation. We do not reproduce it here because it is quite large: it has 41 locations.
- Second, the relations between clocks of the TA and clocks of the game are more precise than the mappings used in [7]. For instance, the relation $x - y = 2$ suffices to express the value of a clock x thanks to a clock y; as another example, one can deduce that $x' < 2$ from $y' < 1$ assuming the relation $0 < x' - y' < 1$. The improvement in precision obtained by

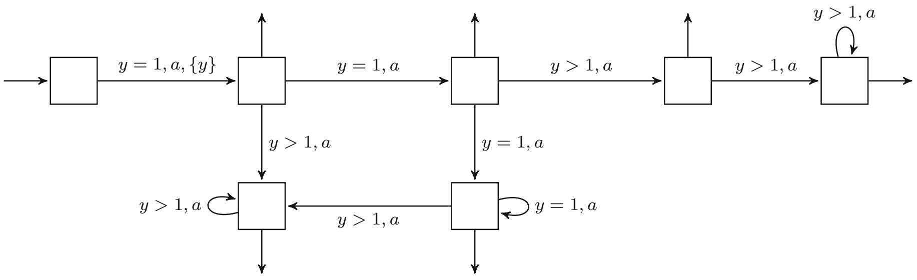
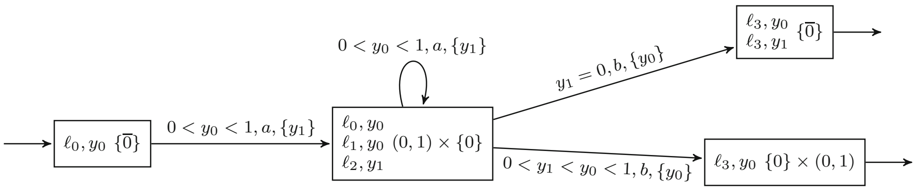


Fig. 11 A deterministic equivalent of the TA of Fig. 10, right



$\ell_0, y_0 \ \{\overline{0}\}$
$0 < y_0 < 1, a, \{y_1\}$
$0 < y_0 < 1, a, \{y_1\}$
ℓ_0, y_0
$\ell_1, y_0 \ (0,1) \times \{0\}$
ℓ_2, y_1
$y_1 = 0, b, \{y_0\}$
ℓ_3, y_0
$\ell_3, y_1 \ \{\overline{0}\}$
$0 < y_1 < y_0 < 1, b, \{y_0\}$
$\ell_3, y_0 \ \{0\} \times (0,1)$

Fig. 12 The result of procedure [7] on our running example

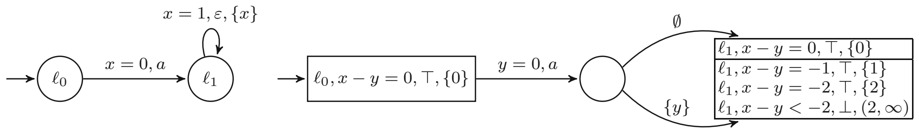
considering relations rather than mappings is sometimes crucial for the determinization. For example, the TA represented on the right-hand side of Fig. 10 can be determinized by our game-approach, but cannot by [7]. Intuitively, the loop in location ℓ_0 forces the procedure of [7] to introduce a new clock at each step of its unfolding, whereas the language remains the same if this loop is removed. A deterministic timed automaton obtained using our prototype using resources (1, 1) for the TA from Fig. 10, right, is depicted on Fig. 11.

Beyond 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 fewer resources. This is the case on the running example of Fig. 1. The deterministic automaton obtained by [7] is depicted in Fig. 12: it needs 2 clocks when our method only needs one.

The same phenomenon happens with timed automata with integer resets. Timed automata with integer resets, introduced in [6], 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 3 *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 run of $\mathcal{A}$, all clocks share the same fractional part. This crucial 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



$x = 1, \varepsilon, \{x\}$
ℓ_0
$x = 0, a$
ℓ_1
$\ell_0, x - y = 0, \top, \{0\}$
$y = 0, a$
$\emptyset$
$\{y\}$
$\ell_1, x - y = 0, \top, \{0\}$
$\ell_1, x - y = -1, \top, \{1\}$
$\ell_1, x - y = -2, \top, \{2\}$
$\ell_1, x - y < -2, \bot, (2, \infty)$

Fig. 13 A timed automaton with ε-transitions and the resulting ε-closure

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 $0 \le c \le M$, 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 3, 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 [7] 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 [14]. Note also that the one-clock timed automaton we obtain coincides with the one obtained by the *ad hoc* determinization of integer reset timed automata [14].

We discussed how our game approach improves the two existing methods for the determinization of timed automata. In the sequel, we define extensions of the game construction in order to deal with invariants and ε-transitions.

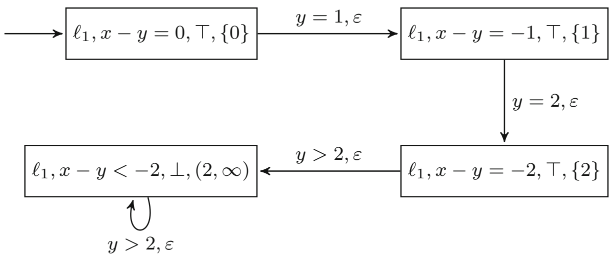
5 Extension to ε-transitions and invariants

In Sect. 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, in order to remove ε-transitions, 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 Fig. 13, left. The resulting game is represented in Fig. 13, right, for resources $(1, 2)$. There, the rightmost state is composed of the state reached without ε-transitions $\{(\ell_1, x - y = 0, \top, \{0\})\}$, and its ε-closure. 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 and two time units. Thus this configuration can only happen when the clock y reaches the region $\{2\}$ or later, whereas the configuration $(\ell_1, x - y = 0, \top)$ could be observed already in region $\{0\}$. More generally, in a given state, a configuration may be associated with several regions, all those being time-successors of the initial region of that state.



$$\begin{tikzcd}
\arrow[r] & \boxed{\ell_1, x-y=0, \top, \{0\}} \arrow[r, "y=1,\varepsilon"] & \boxed{\ell_1, x-y=-1, \top, \{1\}} \arrow[d, "y=2,\varepsilon"] \\
& \boxed{\ell_1, x-y<-2, \bot, (2,\infty)} \arrow[loop below, "y>2,\varepsilon"] & \boxed{\ell_1, x-y=-2, \top, \{2\}} \arrow[l, "y>2,\varepsilon"']
\end{tikzcd}$$

Fig. 14 Step-wise computation of the ε-closure, before merging

Computing the ε-closure of a state in the game amounts to computing the set of reachable configurations by ε-transitions, and associating with every new 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, Fig. 14 represents the computation of the ε-closure discussed above, for a single clock y and maximum constant 2. This alternative point of view justifies that the computation always terminates.

Apart from the structure of the individual states, the set Bad also requires to be redefined when taking into account possible ε-transitions, in particular because regions are now attached to configurations and no longer to states. Intuitively, a state is safe (that is, not in Bad) as soon as, if it contains final configurations, then there exists a final configuration marked $\top$, and corresponding to the initial region in that state. The set Bad should therefore include those states where either the ε-closure leads to a final configuration after some delay or the final configurations with initial region (upper part of the state in the examples) are all marked $\bot$. Indeed, in both cases, the state will be declared as final in the timed automaton $\mathsf{Aut}(\sigma)$ for any fixed strategy σ. However, Determinizator should avoid these states in order to ensure that no over-approximation occurred.

We now come to the formal definition of the game.

Definition 5 Let $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E)$ be a timed automaton and (k, M') resources. We let $\mathbf{M} = \max(M, M')$, and Y a set of k clocks. The game associated with $\mathcal{A}$ and (k, M') is $\mathcal{G}_{\mathcal{A},(k,M')} = (\mathsf{V}, \mathsf{v}_0, \mathsf{Act}, \delta, \mathsf{Bad})$ where:

- $\mathsf{V} = \mathsf{V}_S \sqcup \mathsf{V}_D$ is the finite set of vertices, with $\mathsf{V}_S \subseteq 2^{L\times \mathsf{Rel}_{\mathbf{M}}(X\cup Y)\times\{\top,\bot\}\times \mathsf{Reg}^Y_{M'}}$ and $\mathsf{V}_D \subseteq \mathsf{V}_S \times \mathsf{Reg}^Y_{M'} \times \Sigma$;
- $\mathsf{v}_0 = \mathsf{cl}_\varepsilon(\{(\ell_0, \bigwedge_{z,z'\in X\cup Y} z - z' = 0, \top, \{\overline{0}\})\})$ is the initial vertex, where cl_ε is defined in Eq. (5) below, and $\mathsf{v}_0 \in \mathsf{V}_S$;
- $\mathsf{Act} = \mathsf{Act}_S \sqcup \mathsf{Act}_D$ is the set of possible actions, $\mathsf{Act}_S = \mathsf{Reg}^Y_{M'} \times \Sigma$ and $\mathsf{Act}_D = 2^Y$;
- $\delta = \delta_S \cup \delta_D$ is the transition relation with δ_S and δ_D defined as follows:
  - $\delta_S \subseteq \mathsf{V}_S \times \mathsf{Act}_S \times \mathsf{V}_D$ is the set of edges of the form $\mathcal{E} \xrightarrow{(r',a)} (\mathcal{E}, (r', a))$ if there exists $(\ell, C, b, r) \in \mathcal{E}$ such that $r' \in \overrightarrow{r}$ and there exists $\ell \xrightarrow{g,a,X'} \ell' \in E$ such that condition $\mathsf{C}_{\neq\emptyset}$ is satisfied, that is $[r' \cap C]_{|X} \cap g \neq \emptyset$,
  - $\delta_D \subseteq \mathsf{V}_D \times \mathsf{Act}_D \times \mathsf{V}_S$ is the set of edges of the form $(\mathcal{E}, (r', a)) \xrightarrow{Y'} \mathcal{E}'$ where

$$\mathcal{E}' = \bigcup_{\gamma\in\mathcal{E}} \bigcup_{\gamma'\in \mathsf{Succ}^\varepsilon_e[r',a,Y'](\gamma)} \mathsf{cl}_\varepsilon(\gamma'); \tag{3}$$

- where $\mathsf{Succ}^{\varepsilon}_{e}$ is the elementary successor function

$$\mathsf{Succ}^{\varepsilon}_{e}[r', a, Y'](\ell, C, b, r) = \{(\ell', C', b', r'_{[Y' \leftarrow 0]}) \mid (\ell', C', b') \in \mathsf{Succ}_{e}[r', a, Y'](\ell, C, b)\}, \quad (4)$$

- and $\mathsf{cl}_{\varepsilon}$ is the ε-closure, $\mathsf{cl}_{\varepsilon}(\ell, C, b, r)$ is defined as the smallest fixpoint of the functional

$$X \mapsto (\ell, C, b, r) \cup \bigcup_{(\ell', C', b', r') \in X} \ \bigcup_{r'' \in \overrightarrow{r'}} \mathsf{Succ}^{\varepsilon}_{e}[r'', \varepsilon, \emptyset](\ell', C', b', r'), \quad (5)$$

– $\mathsf{Bad} = \big\{\{(\ell_j, C_j, \perp, r_j)\}_j\big\} \cup \big\{\{(\ell_j, C_j, b_j, r_j)\}_j \mid \forall h \ \big((\cup_j r_j \subseteq \overrightarrow{r_h}) \Rightarrow (\ell_h \in F \Rightarrow b_h = \perp)\big) \wedge (\exists i,\ \ell_i \in F)\big\}$ is the set of bad states.

We now detail the edge relation δ which gives 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 as the state $\mathsf{v}_D = (\{(\ell_j, C_j, b_j, r_j)\}_j, (r', a)) \in \mathsf{V}_D$ provided 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, formally defined above in Eq. (3), 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 defined in Eq. (4) using the basic elementary successor Succ_e defined in Eq. (1) on page 9. To complete this definition, let us discuss 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 the smallest set of configurations containing (ℓ, C, b, r) and closed under elementary successor for any pair (r', ε) where r' is a time successor of the source configuration and without resetting any clocks in Y. It is thus the smallest fixpoint of the functional in Eq. (5). The termination in finite time of the iterative computation of the fixpoint comes from the fact that the number of configurations is finite.

5.2 Invariants

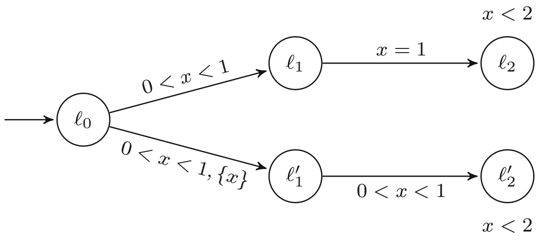
We now explain how to adapt the framework to timed automata with invariants. Note that any timed automaton with invariants can be transformed into an equivalent one without invariants. Yet, this comes with a cost: one obtains a TA in which the guards are regions, and therefore contain diagonal constraints. Also, the size of the resulting automaton may be exponential in the size of the original TA. In our work, we chose to focus on TAs without diagonal guards. One motivation for this is that the forward reachability analysis using the natural extrapolation operator is not correct when applied to TAs with diagonal guards [15].

The adaptation of the game from Definition 5 to TAs with invariants can be summarized as follows. 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 corresponding to the invariants of the original locations. Last, the set of bad states needs to be redefined.

An invariant over clocks of Y is attached to each state of Spoiler. A state v_S of Spoiler thus has the form $\mathsf{v}_S = (\{(\ell_j, C_j, b_j, r_j)\}_j, \mathsf{I})$ where $\mathsf{I} \in I_{M'}(Y)$ is intuitively the most restrictive invariant that over-approximates every invariant for the configurations composing v_S. Formally,

$$\mathsf{I} = \bigcup \big\{r'' \in \mathsf{Reg}^{Y}_{M'} \mid \exists j,\ r'' \in \overrightarrow{r_j} \wedge [r'' \cap C_j]_{|X} \cap \mathsf{Inv}(\ell_j) \neq \emptyset\big\}. \quad (6)$$

Fig. 15 A timed automaton with invariants



In the computation of the successor states, the invariants are taken care of similarly to the guards: their satisfaction is checked on both end-points of the transitions. In order to do so, in the definition of $\mathsf{Succ}_e^\varepsilon$ for a transition $\ell \xrightarrow{g,a,X'} \ell'$ the condition $\mathsf{C}_{\neq\emptyset}$, that is $[r' \cap C]_{|X} \cap g \neq \emptyset$, is replaced with the condition $\mathsf{D}_{\neq\emptyset} := [r' \cap C]_{|X} \cap g \cap \mathsf{Inv}(\ell) \cap (\mathsf{Inv}(\ell'))_{[X' \leftarrow 0]^{-1}} \neq \emptyset$.

The boolean has to take into account the potential over-approximation of the invariant. It is thus redefined as follows:

$$b' = b \wedge \Big([r' \cap C]_{|X} \subseteq (g \cap \mathsf{Inv}(\ell) \cap (\mathsf{Inv}(\ell'))_{[X' \leftarrow 0]^{-1}})\Big) \wedge \\ \times \Big(\big[[\mathsf{Inv}(\ell) \cap C]_{|Y} \cap C\big]_{|X} = \mathsf{Inv}(\ell)\Big) \wedge \Big([\mathsf{Inv}(\ell) \cap C]_{|Y} = \mathsf{I}\Big) \tag{7}$$

where I is the invariant of the state containing this configuration. Indeed, in order to have no over-approximation of the invariant for a configuration, it is necessary that the invariant associated to the state of the game is not larger than the invariant induced by the configuration, moreover this latter should not be an approximation of the invariant in $\mathcal{A}$. As a consequence, the configurations which are built via an approximation of some invariant are marked $\perp$. The relation update is also redefined, to enforce the satisfaction of the invariants:

$$\mathsf{up}(r', C, g, \ell, \ell', X', Y') = \overleftrightarrow{(r' \cap C \cap g \cap \mathsf{Inv}(\ell))_{[X' \leftarrow 0][Y' \leftarrow 0]} \cap \mathsf{Inv}(\ell')}. \tag{8}$$

Note that in this definition, the invariant and the configurations are interdependent. This is no problem. Configurations can be first computed assuming that the last condition $\big([\mathsf{Inv}(\ell) \cap C]_{|Y} = \mathsf{I}\big)$ for b' to be $\top$ is true. Then, I can be computed using configurations and last markers can be updated taking into account I.

Some over-approximation in the invariants can yield a strict over-approximation of the original timed language. The preservation of the property that any winning strategy for Determinizator yields a deterministic equivalent of the original timed automaton is ensured thanks to the booleans in configurations taking into account this risk. The definition of the set Bad is thus unchanged.

Example Let us illustrate the construction of the game, and in particular, the computation of invariants, on an example. Figure 15 represents a timed automaton with invariants. An excerpt of the corresponding game, over resources $(1, 3)$, is depicted in Fig. 16. We consider the right-most state in the picture, and explain how its configurations and its invariant are derived. For the first configuration, with location ℓ_2, the induced invariant is $y < 2$, it is trivially obtained from the invariant $x < 2$ in ℓ_2 and the relation $x - y = 0$. The region over y is simply $y = 1$, and there was no over-approximations so far. The first configuration thus should be $(\ell_2, x - y = 0, \top, \{1\})$. Concerning the configuration with location ℓ_2', the induced invariant is $y < 3$, since $x < 2$ and the relation $0 > x - y > -1$ imply $y < 3$. Note that the region $2 < y < 3$ is necessarily included in this invariant because e.g., the valuation

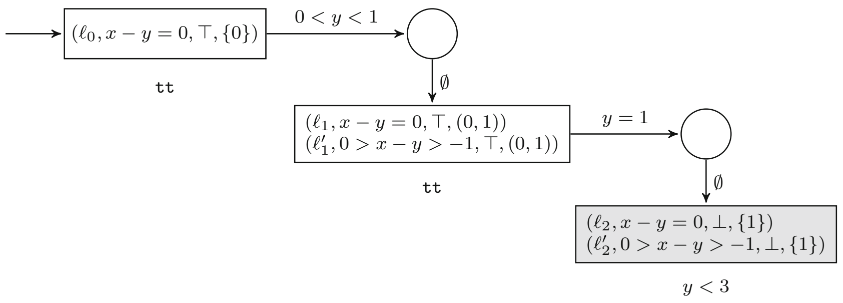

$(\ell_0, x-y=0, \top, \{0\})$
tt
$0<y<1$
$\emptyset$
$(\ell_1, x-y=0, \top, (0,1))$
$(\ell_1', 0>x-y>-1, \top, (0,1))$
tt
$y=1$
$\emptyset$
$(\ell_2, x-y=0, \bot, \{1\})$
$(\ell_2', 0>x-y>-1, \bot, \{1\})$
$y<3$

Fig. 16 Excerpt of the game with resources (1, 3) for the TA from Fig. 15

$x = 1.9$ and $y = 2.1$ satisfies $0 > x - y > -1$, $2 < y < 3$ and $x < 2$. Also, the boolean is $\bot$ since over-approximations occurred in the last step leading to this configuration. The second configuration is thus $(\ell_2', 0 > x - y > -1, \bot, \{1\})$. Last, the invariant associated with the state is the union of the invariants for each configuration $y < 2$ and $y < 3$. It is therefore over-approximated for the first configuration, which explains that its boolean is set to $\bot$ in the end.

5.3 Properties of the strategies in the extended game

Theorem 1, for timed automata with no ε-transitions and no invariants, extends to timed automata with these features, using the extended game defined above. Recall that $\mathsf{Aut}(\sigma)$, the timed automaton derived from the game by fixing a strategy σ for Determinizator, is defined in Definition 4.

Theorem 2 *Let $\mathcal{A}$ be a timed automaton, and (k, M') resources. 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 $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathsf{Aut}(\sigma))$.*

Note that the game construction described in the current section is a conservative extension of the one given in Sect. 3.1 for timed automata with no ε-transitions and no invariants. As a consequence, the following proof of Theorem 2 also serves as a proof for Theorem 1.

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

($\subseteq$): Let σ be any 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 $\mathcal{T}_{\mathsf{Aut}(\sigma)}$ associated with $\mathcal{A}$ and $\mathsf{Aut}(\sigma)$: $\mathcal{T}_{\mathsf{Aut}(\sigma)}$ weak timed simulates $\mathcal{T}_{\mathcal{A}}$. Let $\mathcal{T}_{\mathcal{A}} = (S, s_0, S_F, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \rightarrow_{\mathcal{A}})$, $\mathcal{T}_{\mathsf{Aut}(\sigma)} = (\mathsf{S}', S_0', \mathsf{S}_F', (\mathbb{R}_+ \times \Sigma), \rightarrow_{\mathsf{Aut}(\sigma)})$, and $\mathcal{R} \subseteq S \times \mathsf{S}'$ the following binary relation:

$$\mathcal{R} = \left\{\big((\ell, v), ((\mathcal{E}, \mathsf{I}), v')\big) \mid \exists (\ell, C, b, r) \in \mathcal{E},\ (v, v') \in C \wedge v' \in \overrightarrow{r}\right\}.$$

Let us prove that $\mathcal{R}$ satisfies the four conditions from Definition 2, to be a weak timed simulation. Given that $\mathsf{Aut}(\sigma)$ has no ε-transitions, the fourth condition can be simplified. We will thus prove the following on $\mathcal{R}$:

(1) $(s_0, S_0') \in \mathcal{R}$,
(2) $(s, S') \in \mathcal{R}$ and $s \in S_F$ implies $S' \in \mathsf{S}_F'$,

(3) for all $(s, S') \in \mathcal{R}$, for all $a \in \Sigma$ whenever $s \xrightarrow{a}_{\mathcal{A}} \tilde{s}$, there exists $\tilde{S}' \in \mathsf{S}'$ such that $(\tilde{s}, \tilde{S}') \in \mathcal{R}$ and $S' \xrightarrow{a}_{\mathsf{Aut}(\sigma)} \tilde{S}'$,
(4) for all $(s, S') \in \mathcal{R}$, whenever $s \xrightarrow{\tau_1}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} s_2 \cdots \xrightarrow{\tau_{n-1}}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} s_n \xrightarrow{\tau_n}_{\mathcal{A}} \tilde{s}$, there exists $\tilde{S}' \in \mathsf{S}'$ such that $(\tilde{s}, \tilde{S}') \in \mathcal{R}$ and $S' \xrightarrow{\tau}_{\mathsf{Aut}(\sigma)} \tilde{S}'$ with $\tau = \sum_{i=1}^{n-1} \tau_i$.

(1) The first condition about the initial states is trivially satisfied, by definition of the initial state in the game, and thus the initial location in $\mathsf{Aut}(\sigma)$.
(2) Accepting locations in $\mathsf{Aut}(\sigma)$ are locations in which there is at least one configuration whose location is accepting in $\mathcal{A}$. As a consequence, the second condition is satisfied by $\mathcal{R}$.

Assume now that $s = (\ell_s, v_s)$ and $S' = ((\mathcal{E}_{S'}, \mathsf{I}_{S'}), v_{S'})$ are states of $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{T}_{\mathsf{Aut}(\sigma)}$ respectively such that $(s, S') \in \mathcal{R}$. Then, there exists a configuration $(\ell_s, C, b, r) \in \mathcal{E}_{S'}$ such that v_s and $v_{S'}$ satisfy the relation C, i.e. $(v_s, v_{S'}) \in C$, and $v_{S'} \in \overrightarrow{r}$. Moreover, if $v_s \in \mathsf{Inv}(\ell_s)$, which is true as soon as s is reachable from s_0, then $v_{S'} \in \mathsf{I}_{S'}$. Indeed, if $v_s \in \mathsf{Inv}(\ell_s)$, then the region $r_{S'}$ containing $v_{S'}$ is such that $r_{S'} \in \overrightarrow{r}$ and $(v_s, v_{S'}) \in r_{S'} \cap C$, hence the condition $\mathsf{D}_{\neq\emptyset}$ is satisfied because $v_s \in [r_{S'} \cap C]_{|X} \cap \mathsf{Inv}(\ell_s)$. Therefore, by definition of $\mathsf{I}_{S'}$ (see Eq. (6)), $r_{S'} \subseteq \mathsf{I}_{S'}$ and thus $v_{S'} \in \mathsf{I}_{S'}$.

(3) Let us prove that the third condition is satisfied. Let $a \in \Sigma$ such that $s \xrightarrow{a}_{\mathcal{A}} \tilde{s}$. This transition comes from some edge $(\ell_s, g, a, X', \ell_{\tilde{s}})$ in $\mathcal{A}$ where $\ell_{\tilde{s}}$ is the location of $\tilde{s}$. Let $r_{S'}$ be the region containing $v_{S'}$. Then, $(v_s, v_{S'}) \in C$ implies that the condition $\mathsf{D}_{\neq\emptyset}$ is satisfied, because $v_s \in [r_{S'} \cap C]_{|X} \cap g \cap \mathsf{Inv}(\ell_s) \cap (\mathsf{Inv}(\ell_{\tilde{s}}))_{[X' \leftarrow 0]^{-1}}$. Hence, $\mathsf{Succ}_e^{\varepsilon}[r_{S'}, a, Y'](\ell_s, C, b, r)$ is not empty (whatever is $Y' \subseteq Y$). As a consequence, by definition of the game, there exists an edge $(\mathcal{E}_{S'}, \mathsf{I}_{S'}) \xrightarrow{r_{S'}, a, Y'} (\mathcal{E}_{\tilde{S}'}, \mathsf{I}_{\tilde{S}'})$ in $\mathsf{Aut}(\sigma)$, with some $Y' \subseteq Y$, and there exists a configuration $(\ell_{\tilde{s}}, C', b', r_{S'[Y' \leftarrow 0]}) \in \mathcal{E}_{\tilde{S}'}$ which is an elementary successor of (ℓ_s, C, b, r). Letting $\tilde{S}' = ((\mathcal{E}_{\tilde{S}'}, \mathsf{I}_{\tilde{S}'}), v_{\tilde{S}'})$ where $v_{\tilde{S}'} = v_{S'[Y' \leftarrow 0]}$, we observe that $(v_{\tilde{s}}, v_{\tilde{S}'}) \in C'$ using the definition of the updates for the relation (Eq. (8) on page 24). Hence $(\tilde{s}, \tilde{S}') \in \mathcal{R}$, which proves condition (3) for $\mathcal{R}$.
(4) Finally, let us prove that $\mathcal{R}$ satisfies the last condition. Consider $s \xrightarrow{\tau_1}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} s_2 \cdots \xrightarrow{\tau_{n-1}}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} s_n \xrightarrow{\tau_n}_{\mathcal{A}} \tilde{s}$, a sequence of delays and ε-transitions from s in $\mathcal{A}$. Letting $s_j = (\ell_j, v_j)$ (for $1 \leq j \leq n$) and $s = s_1$, for every $1 \leq j \leq n-1$ there exists an edge in $\mathcal{A}$ of the form $(\ell_j, g_j, \varepsilon, X_j, \ell_{j+1})$ with $v_j + \tau_j \models g_j \cap \mathsf{Inv}(\ell_j) \cap (\mathsf{Inv}(\ell_{j+1}))_{[X_j \leftarrow 0]^{-1}}$ and $v_{j+1} = (v_j + \tau_j)_{[X_j \leftarrow 0]}$. By definition of the ε-closure operator cl_ε (Eq. (5) on page 23), for each index $1 \leq j \leq n$ there is a configuration $(\ell_j, C_j, b_j, r_j) \in \mathcal{E}_{S'}$ such that $(v_j, v_{S'} + \sum_{i=1}^{j-1} \tau_i) \in C_j$ and $v_{S'} + \sum_{i=1}^{j-1} \tau_i \in r_j$. As a consequence, $((\ell_j, v_j), ((\mathcal{E}_{S'}, \mathsf{I}_{S'}), v_{S'} + \sum_{i=1}^{j-1} \tau_i)) \in \mathcal{R}$ and, since the invariant $\mathsf{Inv}(\ell_j)$ is satisfied by v_j, we get that $v_{S'} + \sum_{i=1}^{j-1}$ satisfies $\mathsf{I}_{S'}$. In particular, this is true for $j = n$. Hence, $(v_n + \tau_n, v_{S'} + \sum_{i=1}^{n} \tau_i) \in C_n$ and $v_{\tilde{s}} = v_n + \tau_n$. Then, letting $\tilde{S}' = ((\mathcal{E}_{\tilde{S}'}, \mathsf{I}_{\tilde{S}'}), v_{S'} + \sum_{i=1}^{n-1} \tau_i)$, we obtain that condition (4) is satisfied by $\mathcal{R}$.

This concludes the proof that $\mathsf{Aut}(\sigma)$ weakly timed simulates $\mathcal{A}$, which implies the language inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathsf{Aut}(\sigma))$.

($\supseteq$): Assume now that σ is a winning strategy in $\mathcal{G}_{\mathcal{A},(k,M')}$. Let us prove that $\mathcal{L}(\mathsf{Aut}(\sigma)) \subseteq \mathcal{L}(\mathcal{A})$.

Let $w \in \mathcal{L}(\mathsf{Aut}(\sigma))$. Then, there exists a run $\rho'_w = S'_0 \xrightarrow{w}_{\mathsf{Aut}(\sigma)} S'_n$ in $\mathsf{Aut}(\sigma)$, such that S'_n is accepting. We want to prove that w also belongs to $\mathcal{L}(\mathcal{A})$. To do so, we first build a configuration path going from the initial configuration $(\ell_0, \bigwedge_{z,z' \in X \cup Y} z - z' = 0, \top, \{\overline{0}_Y\})$

to a configuration of S'_n whose location is accepting. This is performed backwards, using the definition of the function $\mathsf{Succ}_e^{\varepsilon}$ for elementary successors. By Eq. (4), this configuration sequence corresponds to a path in $\mathcal{A}$ (that is, an alternating sequence of locations and edges of $\mathcal{A}$). Then we show that along this path in $\mathcal{A}$, there exists a run of $\mathcal{A}$ reading w. This will allow us to conclude that $w \in \mathcal{L}(\mathcal{A})$ and thus $\mathcal{L}(\mathsf{Aut}(\sigma)) \subseteq \mathcal{L}(\mathcal{A})$.

Before the proof of these two steps, we introduce some notations. The accepting run over w in $\mathsf{Aut}(\sigma)$ is $\rho'_w = S'_0 \xrightarrow{\tau_0}_{\mathsf{Aut}(\sigma)} \xrightarrow{a_1}_{\mathsf{Aut}(\sigma)} S'_1 \cdots S'_{n-1} \xrightarrow{\tau_{n-1}}_{\mathsf{Aut}(\sigma)} \xrightarrow{a_n}_{\mathsf{Aut}(\sigma)} S'_n$ with $w = (\sum_{l=0}^{j-1} \tau_l, a_j)_{1 \le j \le n}$. We further write $S'_i = (L_{S'_i}, v_{S'_i})$ for all $0 \le i \le n$ with $L_{S'_i} = (\mathcal{E}'_i, \mathsf{I}'_i)$ and denote by $\ell_\gamma, C_\gamma, b_\gamma$ and r_γ respectively the location, the relation, the boolean and the region of a configuration γ.

5.3.1 Construction of a path $\pi in \mathcal{A}$

Let j be a fixed index such that $1 \le j \le n$. Then, for every configuration in $\mathcal{E}'_j$, one can follow backwards the elementary successors by ε-transitions until a configuration which is the elementary successor of a configuration in $\mathcal{E}'_{j-1}$. Repeating this, one can backwards follow the whole run ρ'_w. Formally, for every configuration $\gamma_j \in \mathcal{E}'_j$ marked $\top$ of S'_j there is a finite sequence $(\gamma_j^i)_{0 \le i \le n_j}$ of configurations marked $\top$ in $L_{S'_j}$ such that:

- there exists $\gamma_{j-1} \in S'_{j-1}$ such that $\gamma_j^0 \in \mathsf{Succ}_e^{\varepsilon}[r^0_{\gamma_j}, a_j, \sigma(L_{S'_{j-1}}, (r_{\gamma_j^0}, a_j))](\gamma_{j-1})$,
- $\gamma_j^{n_j} = \gamma_j$,
- for all $1 \le i \le n_j$, $\gamma_j^i \in \mathsf{Succ}_e^{\varepsilon}[r_{\gamma_j^i}, \varepsilon, \emptyset](\gamma_j^{i-1})$.

Remark that the fact that configurations are marked $\top$ is implied, by definition of $\mathsf{Succ}_e^{\varepsilon}$ (see Eq. (4)), by the fact that γ_j itself is marked $\top$.

We can thus consider the configuration path π, corresponding to the entire run ρ'_w starting the backward construction from an accepting configuration γ_n marked $\top$ in $L_{S'_n}$, because accepting locations of $\mathsf{Aut}(\sigma)$ are states containing at least one configuration whose location is accepting, and by definition of Bad because σ is winning. The path π is thus of the following form:

$$\pi = (\ell_0, \bigwedge_{z,z' \in X \cup Y} z - z' = 0, \top, \{\bar{0}_Y\}) \xrightarrow{\varepsilon} \gamma_0^1 \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \gamma_0^{n_0} \xrightarrow{a_1}\xrightarrow{\varepsilon}$$
$$\cdots \xrightarrow{\varepsilon} \gamma_{j-1}^0 \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \gamma_{j-1}^{n_{j-1}} \xrightarrow{a_j} \gamma_j^0 \xrightarrow{\varepsilon} \cdots \xrightarrow{a_n} \gamma_n.$$

Then, still by definition of the function $\mathsf{Succ}_e^{\varepsilon}$ in Eq. (4), this configuration path corresponds to a path in $\mathcal{A}$. It is thus sufficient to prove that there is a run ρ_π reading w in $\mathcal{A}$ along this path, that is:

$$\rho_\pi = (\ell_0, \{\bar{0}\}) \xrightarrow{\tau_0^0}\xrightarrow{\varepsilon} \left(\ell_{\gamma_0^1}, v_0^1\right) \xrightarrow{\tau_0^1}\xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \left(\ell_{\gamma_0^{n_0}}, v_0^{n_0}\right) \xrightarrow{\tau_0^{n_0}}\xrightarrow{a_1} \left(\ell_{\gamma_1^0}, v_1^0\right) \xrightarrow{\tau_1^0}\xrightarrow{\varepsilon}$$
$$\cdots \xrightarrow{\varepsilon} \left(\ell_{\gamma_{j-1}^0}, v_{j-1}^0\right) \xrightarrow{\tau_{j-1}^0}\xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \left(\ell_{\gamma_{j-1}^{n_{j-1}}}, v_{j-1}^{n_{j-1}}\right) \xrightarrow{\tau_{j-1}^{n_{j-1}}}\xrightarrow{a_j} \cdots \xrightarrow{a_n} \left(\ell_{\gamma_n}, v_n\right)$$

where for all $0 \le j \le n-1$, $\sum_{i=1}^{n_j} = \tau_j$.

5.3.2 Reading w along $\pi in \mathcal{A}$

Let us prove that one can define delays along π to obtain a run in $\mathcal{A}$ reading w. We even prove a stronger fact: for each fragment of the path corresponding to one transition of the run ρ'_w in

$\mathsf{Aut}(\sigma)$, from any valuation $v \in \mathbb{R}_+^X$ in relation with the valuation $v_{S'_{j-1}} \in \mathbb{R}_+^Y$, one can define suitable delays. Formally, let us prove that for every $1 \leq j \leq n$, if $(v^0_{j-1}, v_{S'_{j-1}}) \in C_{\gamma^0_{j-1}}$ then there are delays (τ^i_{j-1})'s such that $(\ell_{\gamma^0_{j-1}}, v^0_{j-1}) \xrightarrow{\tau^0_{j-1}} \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} (\ell_{\gamma^{n_{j-1}}_{j-1}}, v^{n_{j-1}}_{j-1}) \xrightarrow{\tau^{n_{j-1}}_{j-1}} \xrightarrow{a_j} (\ell_{\gamma^0_j}, v^0_j)$ is a run of $\mathcal{A}$ with $\sum_{i=1}^{n_j} \tau^i_{j-1} = \tau_j$ and $(v^0_j, \gamma^0_j) \in C_{\gamma^0_j}$. Observe that this property holds for $j = 1$ since $(\ell_0, \{\overline{0}_X\})$ corresponds to $(\ell_0, \overrightarrow{\{\overline{0}_{X\cup Y}\}}, \top, \{\overline{0}_Y\})$, formally $(\{\overline{0}_X\}, \{\overline{0_Y}\}) \in \overrightarrow{\{\overline{0}_{X\cup Y}\}}$.

The proof is structured as follows. We first show that invariants of $\mathcal{A}$ are satisfied in all the states corresponding to configurations of the path. Then we prove that transition a_j can be fired in states of $\mathcal{A}$ corresponding to the configuration $\gamma^{n_{j-1}}_{j-1}$ (with the associated valuation in ρ'_w) and reach a state corresponding to γ^0_j. Finally we explain how to define delays in such a way that from any state corresponding to γ^0_{j-1}, one reaches a state corresponding to $\gamma^{n_{j-1}}_{j-1}$.

- *Invariants* assuming that $(v^i_{j-1}+\tau^i_{j-1}, v_{S'_{j-1}} + \sum_{h=0}^{i} \tau^h_{j-1}) \in C_{\gamma^i_{j-1}}$, we prove that $v^i_{j-1} + \tau^i_{j-1} \in \mathsf{Inv}(\ell_{\gamma^i_{j-1}})$. Indeed, as γ^i_{j-1} is marked $\top$, $\big[[\mathsf{Inv}(\ell_{\gamma^i_{j-1}}) \cap C_{\gamma^i_{j-1}}]_{|Y} \cap C_{\gamma^i_{j-1}}\big]_{|X} = \mathsf{Inv}(\ell_{\gamma^i_{j-1}})$ and $[\mathsf{Inv}(\ell_{\gamma^i_{j-1}}) \cap C_{\gamma^i_{j-1}}]_{|Y} = \mathsf{I}'_{j-1}$ by definition of I'_{j-1} (Eq. (6) on page 23). As a consequence $v_{S'_{j-1}} + \sum_{h=0}^{i} \tau^h_{j-1} \in \mathsf{I}'_{j-1}$ implies $v^i_{j-1} + \tau^i_{j-1} \in \mathsf{Inv}(\ell_{\gamma^i_{j-1}})$. In words, assuming that valuations in $\mathcal{A}$ satisfy corresponding relations with corresponding valuations in $\mathsf{Aut}(\sigma)$, invariants of locations of $\mathcal{A}$ are satisfied.
- *Discrete transitions labeled in Σ* let $(\ell_{\gamma^{n_{j-1}}_{j-1}}, v^{n_{j-1}}_{j-1} + \tau^{n_{j-1}}_{j-1}) \xrightarrow{a_j} (\ell_{\gamma^0_j}, v^0_j)$ be a discrete transition of ρ_π, labeled in Σ. Assuming that $(v^{n_{j-1}}_{j-1} + \tau^{n_{j-1}}_{j-1}, v_{S'_{j-1}} + \tau_{j-1}) \in C_{\gamma^{n_{j-1}}_{j-1}}$, we prove that it is a transition of $\mathcal{A}$ and that $(v^0_j, v_{S'_j}) \in C_{\gamma^0_j}$.
  By construction of ρ_π, $\gamma^0_j \in \mathsf{Succ}^\varepsilon_e[r_j, a_j, Y_j](\gamma^{n_{j-1}}_{j-1})$. Moreover, by definition of ρ'_w, $v_{S'_{j-1}} + \tau_{j-1} \in r_j$. Then, by definition of $\mathsf{Succ}^\varepsilon_e$ in Sect. 5.2, and because γ^0_j is marked $\top$, there exists an edge $(\ell_{\gamma^{n_{j-1}}_{j-1}}, g_j, a_j, X_j, \ell_{\gamma^0_j})$ in $\mathcal{A}$ such that conditions $\mathsf{D}_{\neq\emptyset}$ and $\mathsf{D}_{\subseteq}$ are satisfied, i.e. $\emptyset \neq [r_j \cap C_{\gamma^{n_{j-1}}_{j-1}}]_{|X} \subseteq g_j \cap \mathsf{Inv}(\ell_{\gamma^{n_{j-1}}_{j-1}}) \cap (\mathsf{Inv}(\ell^0_j))_{[X_j \leftarrow 0]^{-1}}$, and $C_{\gamma^0_j} = \mathsf{up}(r_j, C_{\gamma^{n_{j-1}}_{j-1}}, g_j, \ell_{\gamma^{n_{j-1}}_{j-1}}, \ell_{\gamma^0_j}, X_j, Y_j)$ (defined in Eq. (8) on page 24). As a consequence, this edge is fireable from $(\ell_{\gamma^{n_{j-1}}_{j-1}}, v^{n_{j-1}}_{j-1} + \tau^{n_{j-1}}_{j-1})$, indeed $v^{n_{j-1}}_{j-1} + \tau^{n_{j-1}}_{j-1} \in [v_{S'_{j-1}} + \tau_{j-1} \cap C_{\gamma^{n_{j-1}}_{j-1}}]_{|X} \subseteq [r_j \cap C_{\gamma^{n_{j-1}}_{j-1}}]_{|X} \subseteq g_j \cap \mathsf{Inv}(\ell_{\gamma^{n_{j-1}}_{j-1}}) \cap (\mathsf{Inv}(\ell^0_j))_{[X_j \leftarrow 0]^{-1}}$. Finally, $(v^0_j, v_{S'_j}) \in C_{\gamma^0_j}$ by definition of up (Eq. (8)).
- *Delays and ε-transitions* let $(\ell_{\gamma^0_{j-1}}, v^0_{j-1}) \xrightarrow{\tau^0_{j-1}} \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} (\ell_{\gamma^{n_{j-1}}_{j-1}}, v^{n_{j-1}}_{j-1}) \xrightarrow{\tau^{n_{j-1}}_{j-1}} (\ell_{\gamma^{n_{j-1}}_{j-1}}, v^{n_{j-1}}_{j-1} + \tau^{n_{j-1}}_{j-1})$ be a sequence of delays and ε-transitions of ρ_π corresponding to the delay τ_{j-1} of ρ'_w in $\mathsf{Aut}(\sigma)$. Assuming that $(v^0_{j-1}, v_{S'_{j-1}}) \in C_{\gamma^0_{j-1}}$, we prove that one can fix τ^i_{j-1}'s such that this is a sequence of transitions of $\mathcal{A}$, $\sum_{h=0}^{n_{j-1}-1} \tau^h_{j-1} = \tau_{j-1}$ and $(v^{n_{j-1}}_{j-1} + \tau^{n_{j-1}}_{j-1}, v_{S'_{j-1}} + \tau_{j-1}) \in C_{\gamma^{n_{j-1}}_{j-1}}$. Note that $v_{S'_{j-1}} \in r_{\gamma^{n_{j-1}}_{j-1}}$ and $v_{S'_{j-1}} + \tau_j \in$

$r_j \subseteq \overrightarrow{r_{\gamma_{j-1}^{n_{j-1}}}}$. Let us define τ_{j-1}^i as follows: $\tau_{j-1}^{n_{j-1}} = \tau_j - \sum_{i=0}^{n_{j-1}-1} \tau_{j-1}^i$ and for all $0 \leq i < n_{j-1}$, $\tau_{j-1}^i = 0$ if $v_{S'_{j-1}} + \sum_{h=0}^{i-1} \tau_{j-1}^h \in r_{j-1}^i$, otherwise we fix τ_{j-1}^i as any delay such that $v_{S'_{j-1}} + \sum_{h=0}^{i} \tau_{j-1}^h \in r_{j-1}^i$ and $v_{S'_{j-1}} + \sum_{h=0}^{i} \tau_{j-1}^h \leq \tau_{j-1}$.

Let us prove by induction over i that for all $0 \leq i < n_{j-1}$, $(\ell_{\gamma_{j-1}^i}, v_{j-1}^i) \xrightarrow{\tau_{j-1}^i} \xrightarrow{\varepsilon} (\ell_{\gamma_{j-1}^{i+1}}, v_{j-1}^{i+1})$ is a transition of $\mathcal{A}$ and $(v_{j-1}^{i+1}, v_{S'_{j-1}} + \sum_{h=0}^{i} \tau_{j-1}^h) \in C_{\gamma_{j-1}^{i+1}}$.
First of all, we initialize thanks to the assumption $(v_{j-1}^0, v_{S'_{j-1}}) \in C_{\gamma_{j-1}^0}$.
Let us fix $0 \leq i < n_{j-1}$ and assume that $(v_{j-1}^i + \tau_{j-1}^i, v_{S'_{j-1}} + \sum_{h=0}^{i} \tau_{j-1}^h) \in C_{\gamma_{j-1}^i}$. Hence $(v_{j-1}^i + \tau_{j-1}^i, v_{S'_{j-1}} + \sum_{h=0}^{i+1} \tau_{j-1}^h) \in C_{\gamma_{j-1}^i}$. Then, we can conclude about the inductive step in the same way as in the previous step for a_j's.
We obtain that it is a sequence of transitions of $\mathcal{A}$ and that $(v_{j-1}^{n_{j-1}}, v_{S'_{j-1}} + \sum_{h=0}^{n_{j-1}-1} \tau_{j-1}^h) \in C_{\gamma_{j-1}^{n_{j-1}}}$, which implies that $(v_{j-1}^{n_{j-1}} + \tau_{j-1}^{n_{j-1}}, v_{S'_{j-1}} + \tau_{j-1}) \in C_{\gamma_{j-1}^{n_{j-1}}}$. □

5.4 Comparison with [8]

In Sect. 4, we compared our approach with existing methods in the restricted case where timed automata neither have invariants nor ε-transitions. The determinization procedure of [7] does not deal with invariants and ε-transitions. We therefore compare our extended approach only with the over-approximation algorithm of [8].

The models in [8] are 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 rely on the fact that termination can be ensured by some abstraction. Second, the urgency in the model is not preserved by the over-approximation construction: the resulting DTA only contains lazy transitions (intuitively the type lazy over-approximates all kinds of urgency). Note that we classically decided to rather use invariants to model urgency, but our approach could be adapted to the same model as in [8], and would preserve urgency more often, the same way as we do for invariants. These observations underline the benefits of our game-based approach for TAs with invariants and ε-transitions compared to existing work.

6 Beyond over-approximation

In the previous sections, a game-based approach has been presented to yield a deterministic over-approximation of a given timed automaton. Yet, we advocate that over-approximations are not always appropriate, and, depending on the context, under-approximations or other approximations might be more suitable. We therefore explain in this section how to adapt our framework in order to generate deterministic under-approximations, and also combine over- and under-approximations.

6.1 Under-approximation

One motivation for building deterministic under-approximations of a regular timed language is that one can decide whether the timed language is approximated provided that the 'largest' language is recognized by a deterministic timed automaton. Therefore, given $\mathcal{A}$

a non-deterministic timed automaton, for every deterministic under-approximation $\mathcal{B}$, one can decide whether the approximation is strict or not, that is whether the reverse inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ also holds. Contrary to what happens for over-approximations, one would thus be able to detect if a losing strategy yet yields a deterministic equivalent to the original timed automaton.

We now briefly explain how to modify the game construction, so that any strategy yields an under-approximation, and any winning strategy provides a deterministic equivalent. When we aim at building an over-approximation, during the construction of the game, all litigious successors (i.e. configurations marked $\perp$) are built, possibly introducing more behaviors than in the original TA. In order to obtain an under-approximation, the litigious successors are simply not constructed. Also, ε-transitions and invariants are not more difficult to handle: (1) the ε-closure is under-approximated (by avoiding to build configurations marked $\perp$); (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. We do not give the complete details of this construction, since in the next subsection we present an extension that subsumes both over-approximations and under-approximations.

6.2 Combining over- and under-approximation

Beyond over-approximations and under-approximations, combinations of both can be meaningful in some contexts. Model-based testing is an example of such contexts. Given a non-deterministic timed automaton $\mathcal{A}$, it can be proved that a deterministic timed automaton $\mathcal{B}$ which over-approximates outputs and under-approximates inputs, preserves the conformance relation *tioco*. As a consequence, test cases can be generated from $\mathcal{B}$, as sound test cases for $\mathcal{B}$ remain sound for $\mathcal{A}$. Details on the application of deterministic approximations to the test generation from nondeterministic timed automata models can be found in [10,16].

Let us now explain more generally how to combine over-approximations 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. Note that (Σ_1, Σ_2)-refinement is defined here only for a pair of timed automata $(\mathcal{A}, \mathcal{A}')$ when $\mathcal{A}'$ has no ε-transition.

First of all, let us introduce some notations to shorten the definition. For any timed word w, we write $s_0 \overset{w}{\Longrightarrow} s$ if there exists a run from s_0 to s reading w. If s is left implicit, we write $s_0 \overset{w}{\Longrightarrow}$, thus meaning that there exists a state s such that $s_0 \overset{w}{\Longrightarrow} s$. We also use this convention for the transition relation $\rightarrow$.

Definition 6 Let $\mathcal{A}$ be a timed automaton and $\mathcal{A}'$ be a timed automaton without ε-transitions over the alphabet $\Sigma = \Sigma_1 \sqcup \Sigma_2$, and $\mathcal{T}_\mathcal{A} = (S, s_0, S_F, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \rightarrow_\mathcal{A})$, $\mathcal{T}_{\mathcal{A}'} = (S', s_0', S_F', (\mathbb{R}_+ \times \Sigma), \rightarrow_{\mathcal{A}'})$ their associated transition systems. We say that $\mathcal{A}$ (Σ_1, Σ_2)-*refines* $\mathcal{A}'$ and write $\mathcal{A} \preceq \mathcal{A}'$ when:

1. if $s_0 \overset{w}{\Longrightarrow}_\mathcal{A} \xrightarrow{\tau_0}_\mathcal{A} \xrightarrow{\varepsilon}_\mathcal{A} \cdots \xrightarrow{\tau_{n-1}}_\mathcal{A} \xrightarrow{\varepsilon}_\mathcal{A} \xrightarrow{\tau_n}_\mathcal{A}$ with $\tau_i \in \mathbb{R}_+$ $(0 \leq i \leq n)$, and $s_0' \overset{w}{\Longrightarrow}_{\mathcal{A}'}$ then $s_0' \overset{w}{\Longrightarrow}_{\mathcal{A}'} \xrightarrow{\tau}_{\mathcal{A}'}$ with $\tau = \sum_{i=0}^{n} \tau_i$;
2. if $s_0 \xRightarrow{w.(t,a_2)}_\mathcal{A}$ where $a_2 \in \Sigma_2$, and $s_0' \overset{w}{\Longrightarrow}_{\mathcal{A}'}$ then $s_0' \xRightarrow{w.(t,a_2)}_{\mathcal{A}'}$;
3. if $s_0' \xRightarrow{w.(t,a_1)}_{\mathcal{A}'}$ where $a_1 \in \Sigma_1$, and $s_0 \overset{w}{\Longrightarrow}_\mathcal{A} \xrightarrow{\tau_0}_\mathcal{A} \xrightarrow{\varepsilon}_\mathcal{A} \cdots \xrightarrow{\tau_{n-1}}_\mathcal{A} \xrightarrow{\varepsilon}_\mathcal{A} \xrightarrow{\tau_n}_\mathcal{A}$ with $\tau_i \in \mathbb{R}_+$ $(0 \leq i \leq n)$ and the accumulated delay in w is $t - \sum_{i=0}^{n} \tau_i$, then $s_0 \xRightarrow{w.(t,a_1)}_\mathcal{A}$.

Intuitively, the (Σ_1, Σ_2)-refinement is in the spirit of the alternating simulation from [17], with timing aspects. Apart from time, the notable difference is that (Σ_1, Σ_2)-refinement is defined at a language level and is not a binary relation between states. Roughly speaking, the link between refinement and alternating simulation is the same as between language inclusion and simulation.

Let us explain the three properties of the definition. The first property specifies that if a given word can be read in both timed automata and that a delay τ can be observed in $\mathcal{A}$ (through possible ε-transitions), then such a delay τ can also be observed in $\mathcal{A}'$. No ε-transitions are allowed in $\mathcal{A}'$ for readability, but a natural extension of this definition can be easily written allowing them. The second property states that if a given word can be read in $\mathcal{A}$ and $\mathcal{A}'$ can read this word except up to the last action, and if this last action belongs to Σ_2, then $\mathcal{A}'$ should be able to read the complete word. These two properties thus express that $\mathcal{A}'$ simulates $\mathcal{A}$, at a language level for Σ_2-actions and delays. Last, the third requirement states the simulation of $\mathcal{A}'$ by $\mathcal{A}$ at a language level for Σ_1-actions.

Remark that even if there is no ε-transitions in $\mathcal{A}$, the definition is not symmetric: $\mathcal{A}$ (Σ_1, Σ_2)-refines $\mathcal{A}'$ does not imply that $\mathcal{A}'$ (Σ_2, Σ_1)-refines $\mathcal{A}$, due to the way delays are taken care of. Our targetted application to test selection is responsible for this choice. In particular, if Σ_1 and Σ_2 consist respectively of the input and output alphabets, the (Σ_1, Σ_2)-refinement relation generalizes the io-refinement relation between deterministic timed automata introduced in [18], which was inspired by the alternating simulation [17]. Then, the inverse relation (which we refer to as generalized io-abstraction) still preserves the *tioco* conformance relation [8]: 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 which is sound for a given specification is also sound for any io-refinement of the specification.

Our goal here is to combine over- and under-approximations in the construction of the game so that any strategy for Determinizator yields a deterministic (Σ_1, Σ_2)-abstraction of the original automaton. The game construction is adapted: transitions over actions of Σ_2 and invariants are over-approximated, whereas transitions over actions of Σ_1 are under-approximated. The definition of the (Σ_1, Σ_2)-refinement imposes global, i.e. language-based, conditions. As a consequence, when performing an over-approximation, information about configurations which are removed by an under-approximation must be kept. Moreover, to deal with ε-transitions, the ε-closure should be over-approximated before a Σ_2-action, and under-approximated before a Σ_1-action. As a consequence, the structure of the states of Spoiler is enriched as follows. The set of configurations is replaced by a pair of sets $(\mathcal{E}^1, \mathcal{E}^2)$ where $\mathcal{E}^1$ is the over-approximation of the set of configurations, and $\mathcal{E}^2$ is the set of configurations built after successive over- and under-approximations, depending on the moves leading to its construction to which we applied the under-approximating ε-closure. The invariant associated with a state of Spoiler is then defined in the same way as before, using the first set of configurations (the over-approximation). Formally, given a state of Spoiler whose first set of configurations is $\{(\ell_j, C_j, b_j, r_j)\}_j$, the invariant is defined as follows:

$$\mathsf{I} = \bigcup \left\{ r'' \in \mathsf{Reg}_{M'}^{Y} \mid \exists j,\ r'' \in \overrightarrow{r_j} \wedge [r'' \cap C_j]_{|X} \cap \mathsf{Inv}(\ell_j) \neq \emptyset \right\}. \tag{9}$$

The over-approximation of the invariants is compatible with under-approximations of some behaviors, 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 too much the guards.

Before giving the formal definition of the game, we introduce the two elementary successor operators (one for over-approximation, the other for under-approximation) as well as the two ε-closure operators. 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' are:

$$\mathsf{Succ}_e^+[r', a, Y'](\ell, C, b, r) = \left\{ \begin{array}{l} (\ell', C', b', r'_{[Y' \leftarrow 0]}) \mid \exists \ell \xrightarrow{g,a,X'} \ell' \in E \text{ such that} \\ \quad [r' \cap C]_{|X} \cap g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}} \neq \emptyset \\ \quad C' = \mathsf{up}(r', C, g, \mathsf{Inv}(\ell), \mathsf{Inv}(\ell'), X', Y') \\ \quad b' = b \wedge ([r' \cap C]_{|X} \subseteq g) \end{array} \right\} \tag{10}$$

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

$$\mathsf{Succ}_e^-[r', a, Y'](\ell, C, b, r) = \left\{ \begin{array}{l} (\ell', C', b', r'_{[Y' \leftarrow 0]}) \mid \exists \ell \xrightarrow{g,a,X'} \ell' \in E \text{ such that} \\ \quad [r' \cap C]_{|X} \subseteq g \cap \mathsf{Inv}(\ell) \cap \quad \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}} \\ \quad C' = \mathsf{up}(r', C, g, \mathsf{Inv}(\ell), \mathsf{Inv}(\ell'), X', Y') \\ \quad b' = b \end{array} \right\} \tag{11}$$

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 $\mathsf{Inv}(\ell')$. The formal definition is given in Eq. (8) page 24.

Roughly, Succ_e^+ yields a set of configurations over-approximating the set of successor states in $\mathcal{A}$. Indeed, successor configurations are built as soon as $\mathsf{D}_{\neq\emptyset}$ is satisfied. On the other side, Succ_e^- under-approximates the set of states using the restrictive condition $\mathsf{D}_{\subseteq}$.

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

$$X \mapsto (\ell, C, b, r) \cup \bigcup_{\substack{(\ell', C', b', r') \in X \\ r'' \in \overrightarrow{r'}}} \mathsf{Succ}_e^+[r'', \varepsilon, \emptyset](\ell', C', b', r'), \text{ and} \tag{12}$$

$$X \mapsto (\ell, C, b, r) \cup \bigcup_{\substack{(\ell', C', b', r') \in X \\ r'' \in \overrightarrow{r'}}} \mathsf{Succ}_e^-[r'', \varepsilon, \emptyset](\ell', C', b', r'), \text{respectively.} \tag{13}$$

We are now in the position to provide the formal definition of the game.

Definition 7 Let $\mathcal{A} = (L, \ell_0, F, \Sigma, X, M, E, \mathsf{Inv})$ be a timed automaton and (k, M') resources. We let $\mathbf{M} = \max(M, M')$, and Y a set of k clocks. The game associated with $\mathcal{A}$ and (k, M') is $\mathcal{G}_{\mathcal{A},(k,M')} = (\mathsf{V}, \mathsf{v}_0, \mathsf{Act}, \delta, \mathsf{Bad})$ where:

– $\mathsf{V} = \mathsf{V}_S \sqcup \mathsf{V}_D$ is a finite set of vertices, made of the disjoint union of the set $\mathsf{V}_S \subseteq (2^{L \times \mathsf{Rel}_{\mathbf{M}}(X \cup Y) \times \{\top, \bot\} \times \mathsf{Reg}_{M'}^Y})^2 \times I_{M'}(Y)$ of vertices of Spoiler, and the set $\mathsf{V}_D \subseteq \mathsf{V}_S \times \mathsf{Reg}_{M'}^Y \times \Sigma$ of vertices of Determinizator.

– $\mathsf{v}_0 = ((\mathsf{cl}^+_\varepsilon, \mathsf{cl}^-_\varepsilon)(\{(\ell_0, \bigwedge_{z,z'\in X\cup Y} z - z' = 0, \top, \{\bar{0}\})\}), \mathsf{I}_0)$ with I_0 the invariant from Eq. (9), is the initial vertex and belongs to player Spoiler;
– Act is the set of possible actions partitioned into $\mathsf{Act}_S = \mathsf{Reg}^Y_{M'} \times \Sigma$ and $\mathsf{Act}_D = 2^Y$;
– $\delta = \delta_S \cup \delta_D$ is the transition relation with δ_S and δ_D defined as follows.

  – $\delta_S \subseteq \mathsf{V}_S \times \mathsf{Act}_S \times \mathsf{V}_D$ is the set of edges of the form $\mathsf{v}_S \xrightarrow{(r',a)} (\mathsf{v}_S, (r', a))$ for $\mathsf{v}_S = ((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I})$ and
    - if $a \in \Sigma_1$ and one of the two following conditions is satisfied
      · $\exists(\ell, C, \top, r) \in \mathcal{E}^2, r' \in \overrightarrow{r}, \exists \ell \xrightarrow{g,a,X'} \ell' \in E$ such that $\mathsf{D}_\subseteq$ is satisfied, i.e. $[r' \cap C]_{|X} \subseteq g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X'\leftarrow 0]^{-1}}$,
      · $\forall(\ell, C, b, r) \in \mathcal{E}^1,\ r' \in \overrightarrow{r}, \exists \ell \xrightarrow{g,a,X'} \ell' \in E$ such that the condition $\mathsf{D}_\subseteq$ is satisfied, i.e. $[r' \cap C]_{|X} \subseteq g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X'\leftarrow 0]^{-1}}$; or
    - if $a \in \Sigma_2$ and $\exists(\ell, C, b, r) \in \mathcal{E}^1$ such that $r' \in \overrightarrow{r}$ and $\mathsf{D}_{\neq\emptyset}$ is satisfied, i.e. $\exists \ell \xrightarrow{g,a,X'} \ell' \in E$ s.t. $[r' \cap C]_{|X} \cap g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X'\leftarrow 0]^{-1}} \neq \emptyset$;
  – $\delta_D \subseteq \mathsf{V}_D \times \mathsf{Act}_D \times \mathsf{V}_S$ is the set of edges of the form $\mathsf{v}_D \xrightarrow{Y'} ((\mathcal{E}'^1, \mathcal{E}'^2), \mathsf{I}')$ for $\mathsf{v}_D = (((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I}), (r', a))$ and
    - if $a \in \Sigma_1$ and the target state satisfies the following conditions $\mathcal{E}'^1 = \mathsf{cl}^+_\varepsilon(\cup_{\gamma\in\mathcal{E}^1} \mathsf{Succ}^+_e[r', a, Y'](\gamma))$, $\mathcal{E}'^2 = \mathsf{cl}^-_\varepsilon(\cup_{\gamma\in\mathcal{E}^2} \mathsf{Succ}^-_e[r', a, Y'](\gamma))$ and I' is defined above in Eq. (9); or
    - if $a \in \Sigma_2$ and the target state satisfies the following conditions $\mathcal{E}'^1 = \mathsf{cl}^+_\varepsilon(\cup_{\gamma\in\mathcal{E}^1} \mathsf{Succ}^+_e[r', a, Y'](\gamma))$, $\mathcal{E}'^2 = \mathsf{cl}^-_\varepsilon(\cup_{\gamma\in\mathcal{E}^1} \mathsf{Succ}^+_e[r', a, Y'](\gamma))$ and I' is defined above in Eq. (9);

– $\mathsf{Bad} = \{(((\{(\ell_j, C_j, \bot, r_j)\}_j, \mathcal{E}^2), \mathsf{I})\} \cup \{(((\{(\ell_j, C_j, b_j, r_j)\}_j, \mathcal{E}^2), \mathsf{I}) \mid \forall h\ \big((\forall j,\ r_j \in \overrightarrow{r_h}) \Rightarrow (\ell_h \in F \Rightarrow b_h = \bot)\big) \wedge (\exists i,\ \ell_i \in F)\} \cup \{((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I}) \mid \exists s \in \mathcal{E}^1, a \in \Sigma_1, r'$ and Y' s.t. $\mathsf{Succ}^+_e[r', a, Y'](s) \neq \emptyset \wedge ((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I}) \xrightarrow{(r',a)}\}$ is the set of bad states.

In words, the possible moves of the players are defined as follows. Given $\mathsf{v}_S = ((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I}) \in \mathsf{V}_S$ a state of Spoiler and (r', a) one of his moves, the successor state is defined as a state $\mathsf{v}_D = (\mathsf{v}_S, (r', a)) \in \mathsf{V}_D$. Note that v_D is built only if a condition depending on a is satisfied:

– if $a \in \Sigma_1$, then one wants to under-approximate the behaviors. To force the under-approximation, v_D is defined only if, either there is a configuration marked $\top$ in $\mathcal{E}^2$ from which a can be fired without approximation, or from all the configurations in $\mathcal{E}^1$, a can be fired without approximation.
– when $a \in \Sigma_2$, the goal is to over-approximate, thus v_D is built if there is at least one configuration in $\mathcal{E}^1$ from which a can be fired.

Given $\mathsf{v}_D = (\mathsf{v}_S, (r', a)) \in \mathsf{V}_D$ a state of Determinizator and $Y' \subseteq Y$ one of its moves, the successor state is $\mathsf{v}'_S = ((\mathcal{E}'^1, \mathcal{E}'^2), \mathsf{I}') \in \mathsf{V}_S$ such that $\mathcal{E}'^1$ is the over-approximation of successor configurations of $\mathcal{E}^1$, and $\mathcal{E}'^2$ is the set of successor configurations obtained by successive over-approximations and under-approximations (depending on the actions). In particular, the ε-closure of $\mathcal{E}'^1$ is over-approximated whereas the ε-closure of $\mathcal{E}'^2$ is under-approximated.

Last, in order to preserve the exactness of winning strategies for Determinizator, the set Bad is extended: states obtained before an under-approximation, that is from which some behaviors are cut, are added to Bad. More precisely, any state of Spoiler v_S containing a configuration (ℓ, C, b, r) in the over-approximating set of configurations such that, for

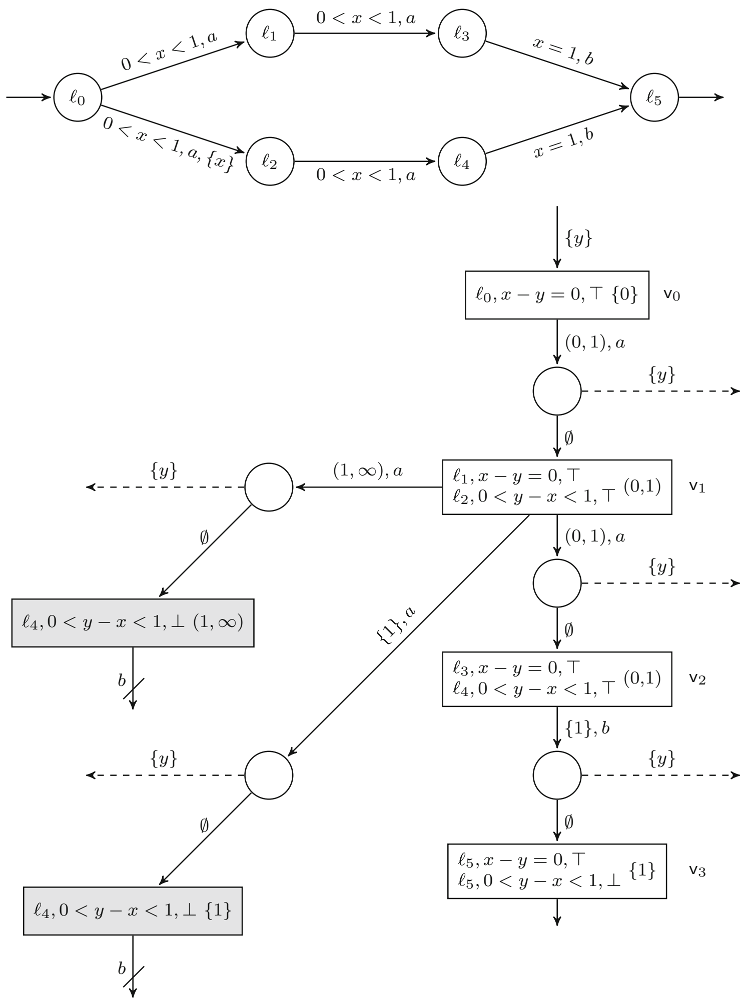


Fig. 17 A timed automaton $\mathcal{A}$ and excerpt of the game $\mathcal{G}_{\mathcal{A},(1,1)}$ with under-approximation for $\Sigma_1 = \{a\}$ and over-approximation for $\Sigma_2 = \{b\}$

(r', a_1) and Y' moves of the two players $\mathsf{Succ}_e^+[r', a_1, Y'](\ell, C, b, r)$ is not empty whereas the successor is not built, is in Bad.

Example An example of a non-deterministic timed automaton is depicted in Fig. 17 with $\Sigma_1 = \{a\}$ and $\Sigma_2 = \{b\}$, with a part of the construction of the associated game $\mathcal{G}_{\mathcal{A},(1,1)}$. For simplicity, we ommitted labels of states of Determinizator in this figure. Also, the labels of states of Spoiler should be understood as follows: the $\mathcal{E}_1$ component contains all configurations, whereas the $\mathcal{E}_2$ component only contains the ones marked $\top$.

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

Proposition 4 *Let $\mathcal{A}$ be a timed automaton over the alphabet $\Sigma = \Sigma_1 \sqcup \Sigma_2$, and (k, M') resources. 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 $\mathcal{A} \preceq \mathsf{Aut}(\sigma)$. Moreover, if σ is winning, then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathsf{Aut}(\sigma))$.*

Proof 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, $\mathcal{A} \preceq \mathsf{Aut}(\sigma)$. Let $\mathcal{T}_{\mathcal{A}} = (S, s_0, S_F, (\mathbb{R}_+ \times (\Sigma \cup \{\varepsilon\})), \rightarrow_{\mathcal{A}})$, $\mathcal{T}_{\mathsf{Aut}(\sigma)} = (\mathsf{S}', S_0'$, S_F', $(\mathbb{R}_+ \times \Sigma), \rightarrow_{\mathsf{Aut}(\sigma)})$, the respective timed transition systems associated with $\mathcal{A}$ and $\mathsf{Aut}(\sigma)$.

Recall the three properties that we have to prove:

1. if $s_0 \overset{w}{\Rightarrow}_{\mathcal{A}} \xrightarrow{\tau_0}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \cdots \xrightarrow{\tau_{n-1}}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \xrightarrow{\tau_n}_{\mathcal{A}}$ with $\tau_i \in \mathbb{R}_+$ $(0 \leq i \leq n)$ and $S_0' \overset{w}{\Rightarrow}_{\mathsf{Aut}(\sigma)}$, then $S_0' \overset{w}{\Rightarrow}_{\mathsf{Aut}(\sigma)} \xrightarrow{\tau}_{\mathsf{Aut}(\sigma)}$ with $\tau = \sum_{i=0}^{n} \tau_i$;
2. if $s_0 \xRightarrow{w.(t,a_2)}_{\mathcal{A}}$ where $a_2 \in \Sigma_2$, and $S_0' \overset{w}{\Rightarrow}_{\mathsf{Aut}(\sigma)}$ then $S_0' \xRightarrow{w.(t,a_2)}_{\mathsf{Aut}(\sigma)}$;
3. if $S_0' \xRightarrow{w.(t,a_1)}_{\mathsf{Aut}(\sigma)}$ where $a_1 \in \Sigma_1$, and $s_0 \overset{w}{\Rightarrow}_{\mathcal{A}} \xrightarrow{\tau_0}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \cdots \xrightarrow{\tau_{n-1}}_{\mathcal{A}} \xrightarrow{\varepsilon}_{\mathcal{A}} \xrightarrow{\tau_n}_{\mathcal{A}}$ with $\tau_i \in \mathbb{R}_+$ $(0 \leq i \leq n)$ and the accumulated delay in w is $t - \sum_{i=0}^{n} \tau_i$, then $s_0 \xRightarrow{w.(t,a_1)}_{\mathcal{A}}$.

Note that the proof of Theorem 2 on page 25 applies to prove the first and the second properties. We define the same binary relation $\mathcal{R} \subseteq S \times \mathsf{S}'$:

$$\mathcal{R} = \left\{\left((\ell, v), ((\mathcal{E}, \mathsf{I}), v')\right) \mid \exists(\ell, C, b, r) \in \mathcal{E},\ (v, v') \in C \wedge v' \in \overrightarrow{r}\right\}.$$

By induction, $s_0 \overset{w}{\Rightarrow}_{\mathcal{A}} s$ and $S_0' \overset{w}{\Rightarrow}_{\mathsf{Aut}(\sigma)} S'$ imply $(s, ((\mathcal{E}^1_{S'}, \mathsf{I}_{S'}), v_{S'})) \in \mathcal{R}$ assuming $S' = (((\mathcal{E}^1_{S'}, \mathcal{E}^2_{S'}), \mathsf{I}_{S'}), v_{S'})$. In words, the state estimate $\mathcal{E}^1_{S'}$ is exactly the contents of the state one would have in the game for over-approximations only.

The heart of the proof thus concerns the third property. In fact, we prove a stronger property:

3′. if $S_0' \xRightarrow{w.(t,a_1)}_{\mathsf{Aut}(\sigma)}$ where $a_1 \in \Sigma_1$, and $s_0 \overset{w}{\Rightarrow}_{\mathcal{A}}$ then $s_0 \xRightarrow{w.(t,a_1)}_{\mathcal{A}}$.

Indeed, we prove that if a timed word ending with an action in Σ_1 can be read in $\mathcal{A}'$ and its largest strict prefix can be read in $\mathcal{A}$, then the entire word can be read in $\mathcal{A}$. Let us assume that $S_0' \xRightarrow{w\cdot(t,a_1)}_{\mathsf{Aut}(\sigma)}$ with $a_1 \in \Sigma_1$, and that $s_0 \overset{w}{\Rightarrow}_{\mathcal{A}}$, and write S' for the state of $\mathsf{Aut}(\sigma)$ such that $S_0' \overset{w}{\Rightarrow}_{\mathsf{Aut}(\sigma)} S'$. Note that S' is unique because $\mathsf{Aut}(\sigma)$ is deterministic. Writing $S' = (((\mathcal{E}^1_{S'}, \mathcal{E}^2_{S'}), \mathsf{I}_{S'}), v_{S'})$, the a_1-transition from S' corresponds to an edge $(((\mathcal{E}^1_{S'}, \mathcal{E}^2_{S'}), \mathsf{I}_{S'}), (r', a_1, Y'), \mathsf{v})$ of $\mathsf{Aut}(\sigma)$. Hence there are two cases. Either (i) there exists $(\ell, C, \top, r) \in \mathcal{E}^2_{S'}$ such that $r' \in \overrightarrow{r}$ and there exists an edge $\ell \xrightarrow{g,a,X'} \ell'$ in $\mathcal{A}$ such that condition $\mathsf{D}_{\subseteq}$ is satisfied, that is $[r' \cap C]_{|X} \subseteq g \cap \mathsf{Inv}(\ell) \cap \mathsf{Inv}(\ell')_{[X' \leftarrow 0]^{-1}}$; or (ii) for all $(\ell, C, b, r) \in \mathcal{E}^1_{S'}$ such that $r' \in \overrightarrow{r}$, there exists an edge $\ell \xrightarrow{g,a,X'} \ell'$ in $\mathcal{A}$ such that condition $\mathsf{D}_{\subseteq}$ holds.

(i) In this case, the second part of the proof of Theorem 2 applies. One can thus build a path in $\mathcal{A}$ along which it is possible to read w, ending in a state of the form $(\ell, \tilde{v})$, such that $(\tilde{v}, v_{S'}) \in C$. As a consequence, $(\tilde{v} + \tau, v_{S'} + \tau) \in C$, where τ is the delay right before a_1 in $w(t, a_1)$, and thus $v_{S'} + \tau \in r'$. Since there exists $\ell \xrightarrow{g,a,X'} \ell'$ in $\mathcal{A}$ such that $\mathsf{D}_{\subseteq}$, $(\ell, \tilde{v} + \tau) \xrightarrow{a_1}_{\mathcal{A}} (\ell', \tilde{v} + \tau_{|X'})$ and in particular invariants of ℓ and ℓ' are satisfied. Hence $s_0 \xRightarrow{w\cdot(t,a_1)}_{\mathcal{A}} (\ell', \tilde{v} + \tau_{[X' \leftarrow 0]})$.

(ii) By assumption, $S_0' \overset{w}{\Rightarrow}_{\mathsf{Aut}(\sigma)} S'$ and $s_0 \overset{w}{\Rightarrow}_{\mathcal{A}} s$, then, as explained above, $(s, ((\mathcal{E}^1_{S'}, \mathsf{I}_{S'}), v_S')) \in \mathcal{R}$. Then, $\mathcal{E}^1_{S'}$ contains a configuration of the form (ℓ, C, b, r) with $s = (\ell, v_s)$, and by (ii), there exists an edge $\ell \xrightarrow{g,a,X'} \ell' \in E$ such that $\mathsf{D}_{\subseteq}$ is satisfied. As a consequence, by the same reasoning as above, $s_0 \xRightarrow{w\cdot(t,a_1)}_{\mathcal{A}} (\ell', v_s + \tau_{[X' \leftarrow 0]})$.

We thus proved that $\mathcal{A} \preceq \mathsf{Aut}(\sigma)$.

Assume now that σ is winning. Thanks to the new definition of the set Bad, in this case we recover the properties of the original method. Indeed, by definition of Bad, for all locations $((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I})$ of $\mathsf{Aut}(\sigma)$, on the one hand, $(\mathcal{E}^1, \mathsf{I})$ is a state of the game built with only over-approximations (see Sect. 5.2), which is not a bad state, and, on the other hand, this state has the same successor as in the original game because of the inclusion

$$\Big\{((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I}) \mid \exists s \in \mathcal{E}^1,\ \exists a \in \Sigma_1,\ \exists r',\ \exists Y' \text{ s.t.}$$
$$\mathsf{Succ}_e^+[r', a, Y'](s) \neq \emptyset \wedge ((\mathcal{E}^1, \mathcal{E}^2), \mathsf{I}) \xrightarrow{(r',a)} \Big\} \subseteq \mathsf{Bad}.$$

Therefore the proof of Theorem 2 applies and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathsf{Aut}(\sigma))$. □

7 Conclusion

In this article, we propose 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 $\mathcal{A}$, in both cases, with k clocks and maximal constant M. We also detail how to adapt the framework 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 [10,16]. Our approach combines the advantages of two existing methods. First, strategies can be seen as a timed variant of the skeletons of [8]: strategies are timed and adaptive, compared to fixed finite-state skeletons. Thus, our game approach yields exact deterministic equivalent TAs on some instances where the over-approximation algorithm from [8] does not. In particular, the game approach preserves deterministic timed automata (when sufficient resources are provided). However, the converse could also be true, due to the faculty of skeletons to unfold the TA. A perspective could then be to combine both approaches. 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 [8] always over-approximates the urgency status of the transitions as lazy.

Compared to the determinization procedure of [7], our approach deals with a richer model of timed automata, including ε-transitions and invariants. Already without these extensions, any timed automaton that can be determinized by [7], can also be determinized by our game-approach. The class of automatically determinized timed automata is thus strictly increased, thanks to a smoother treatment of relations between the original and the new clocks, and also due to a partial treatment of language inclusion between distinct paths of the original automaton.

The (approximate) determinization of timed automata is a complex problem and the three above mentioned algorithms run in time doubly exponential in the size of the input. More precisely for our approach, the number of locations of the resulting automaton is doubly exponential in its number of clocks and in the number of clocks of $\mathcal{A}$, and exponential in the number of locations of $\mathcal{A}$. We implemented a prototype tool during a visit in the team of Kim G. Larsen at Aalborg University and in particular thanks to the help of Peter Bulychev. Given the difficulty of the problem, it would be of interest to develop some heuristics.

First, some timed automata are identified as "easily" determinizable. It is the case of integer reset timed automata, event-clock timed automata or for input-determined timed automata (in which resets depend only on the timed word, not on the run), for which greedy determinization algorithms exist. A preprocessing could thus deal efficiently with these simple cases of non-determinism, and the greedy algorithms could then be used to develop heuristics to choose a good exploration of the game.

Second, the resources and other features of the algorithm could be optimized online. During the on-the-fly construction of the game while searching for a winning strategy, resource clocks could be added if necessary, or the precision of the guards and relations could be increased. For example, relations can be restricted to mapping, zones can be used instead of regions, the maximal constant in the relations and the guards can be modified. The current prototype implementation uses zones rather than regions which limits the size of the game in practice, even if the complexity is unchanged.

Acknowledgments This work was partially funded by the TESTEC Project of the french research agency (ANR).

Appendix

Proof of Proposition 2 Let us assume that Y is totally ordered, and for $Y' \subseteq Y$, we write $\min(Y')$ for the smallest element in Y' according to the total order, with the convention that $\min(\emptyset) = \emptyset$. Given a winning strategy $\sigma : \mathsf{V}_D \to \mathsf{Act}_D$ for Determinizator, we define $\sigma' : \mathsf{V}_D \to Y \cup \{\emptyset\}$, with atomic resets, iteratively. The idea is to simulate σ using only atomic resets. Instead of having several clocks with same value, we reset only one clock and use a mapping, along the construction, to store which clock is used to represent which set of clocks. Then, one considers triples with a state reached following σ', the corresponding state reached following σ and a mapping γ which assigns to each clock, the clock used in the state of σ' to represents its value in the state of σ.

- $\mathsf{Temp} := \{(\mathsf{v}, \mathsf{v}, \mathsf{Id}) \in \mathsf{V}_D \times \mathsf{V}_D \times Y^Y \mid \exists (r, a) \in \mathsf{Act}_S \text{ s.t. } (\mathsf{v}_0, r, a, \mathsf{v}) \in \delta\}$
- $\mathsf{V}_{\mathsf{def}} := \emptyset$
- While $\mathsf{Temp} \neq \emptyset$
  - take $(\mathsf{v}', \mathsf{v}, \gamma)$ in Temp
  - if $\mathsf{v}' \notin \mathsf{V}_{\mathsf{def}}$ then
    - $\sigma'(\mathsf{v}') := \min(\sigma(\mathsf{v}))$
    - $\gamma'(y) := \begin{cases} \sigma'(\mathsf{v}') & \text{if } y \in \sigma(\mathsf{v}), \\ \gamma(y) & \text{otherwise} \end{cases}$
    - $\mathsf{Temp} := \mathsf{Temp} \cup \{(\mathsf{w}', \mathsf{w}, \gamma') \mid \exists \mathsf{v}'_S, \mathsf{v}_S \in \mathsf{V}_S,\ \exists r'' \in \mathsf{Reg}^Y_{M'},\ \exists b \in \Sigma \text{ s.t. } \mathsf{v}' \xrightarrow{\sigma'(\mathsf{v}')} \mathsf{v}'_S \xrightarrow{(r'',b)} \mathsf{w}' \wedge \mathsf{v} \xrightarrow{\sigma(\mathsf{v})} \mathsf{v}_S \xrightarrow{(r''_{[y \leftarrow \gamma'(y)]},b)} \mathsf{w}\ \}$
      Note that by definition of the set of edges of the game, $\mathsf{w} = (\mathsf{v}_S, (r''_{[y \leftarrow \gamma'(y)]}, b))$ and $\mathsf{w}' = (\mathsf{v}'_S, (r'', b))$.
    - $\mathsf{V}_{\mathsf{def}} := \mathsf{V}_{\mathsf{def}} \cup \{\mathsf{v}'\}$
- For all $\mathsf{v}_D \in \mathsf{V}_D \setminus \mathsf{V}_{\mathsf{def}}$
  - $\sigma'(\mathsf{v}_D) = \emptyset$

Intuitively, the above algorithm is a search of $\mathsf{Aut}(\sigma)$. We propagate the encoding of the clocks of σ by clocks in σ' and iteratively build σ'. The set Temp represents the triples to be processed and the set $\mathsf{V}_{\mathsf{def}}$ represents the states of Determinizator in the game for which the strategy σ' is defined. Moreover, the last step of the algorithm consists in arbitrarily defining the strategy for the unvisited states. By construction, these states are not reachable when Determinizator follows σ' hence this choice does not impact. Thus, σ' is well defined. The correction is a bit tedious, but intuitive: relations in the states in σ' give at least as much information as in σ because the time information of each clock x in σ is carried by $\gamma(x)$ in the corresponding state in σ'. The duplication of the information does not help to win.

Let us prove formally that σ' is a winning strategy using the following lemma.

Lemma 1 *For all* $(\mathsf{v}', \mathsf{v}, \gamma) \in \mathsf{V}_{\mathsf{def}}$ *with* $\mathsf{v}' = (\mathcal{E}_{\mathsf{v}'}, (r', a'))$ *and* $\mathsf{v} = (\mathcal{E}_{\mathsf{v}}, (r, a))$*:*

(i) $a' = a$
(ii) $r = r'_{[y \leftarrow \gamma(y)]}$
(iii) v *has no predecessor in* Bad
(iv) $\forall \mathsf{w}'$ *such that* $\exists (\mathsf{v}'_S, r'', b) \in \mathsf{V}_S \times \mathsf{Act}_S$ *with* $\mathsf{v}' \xrightarrow{\sigma'(\mathsf{v}')} \mathsf{v}'_S \xrightarrow{(r'',b)} \mathsf{w}'$, *then* $\exists (\mathsf{w}_0, \gamma_0) \in \mathsf{V}_D \times Y^Y$ *such that* $(\mathsf{w}', \mathsf{w}_0, \gamma_0) \in \mathsf{V}_{\mathsf{def}}$
(v) $\mathcal{E}_{\mathsf{v}} \subseteq \{(\ell, C'_{[y \leftarrow \gamma(y)]}, \top) \mid (\ell, C', \top) \in \mathcal{E}_{\mathsf{v}'}\} \cup \{(\ell, C, \bot)\}$
(vi) $\mathcal{E}_{\mathsf{v}'} \subseteq \{(\ell, C', b') \mid (\ell, C'_{[y \leftarrow \gamma(y)]}, b) \in \mathcal{E}_{\mathsf{v}}\}$

Let us first assume that this lemma is true. The fifth item implies that for all $(\mathsf{v}', \mathsf{v}, \gamma) \in \mathsf{V}_{\mathsf{def}}$, ($\mathsf{v} \notin \mathsf{Bad} \Rightarrow \mathsf{v}' \notin \mathsf{Bad}$). The third item implies that the v''s are not in Bad. The fourth item implies that only these states v' (appearing in $\mathsf{V}_{\mathsf{def}}$) impact on the fact that σ' is winning or not. As a consequence, if Lemma 1 is true, then σ' is winning.

Let us now prove Lemma 1.

Proof of Lemma 1 Items (i), (ii), (iii) are satisfied by triples initially added to Temp. Moreover, the updates of Temp only add triples satisfying (i) and (ii) and whose a predecessor v_S of the second element is a state of Spoiler reachable following σ, hence (iii) is satisfied too (only the configurations of v_S impact on whether $\mathsf{v}_S \in \mathsf{Bad}$ or not, and all the predecessors of v share the same set of configurations, by definition of δ). Therefore, items (i), (ii), (iii) are satisfied for all triples of $\mathsf{V}_{\mathsf{def}}$.

Let us prove by induction that items (v) and (vi) are satisfied for all triples $(\mathsf{v}', \mathsf{v}, \gamma) \in \mathsf{V}_{\mathsf{def}}$. Remark that all triples in $\mathsf{V}_{\mathsf{def}}$ are first added to Temp. Triples initially added to Temp are such that $\mathcal{E}_{\mathsf{v}} = \mathcal{E}_{\mathsf{v}'}$, hence they satisfy (v) and (vi). Let us prove now that if $(\mathsf{v}', \mathsf{v}, \gamma) \in \mathsf{V}_{\mathsf{def}}$ satisfies (v) and (vi), then triples added to Temp during the inner loop for this triple, satisfy (v) and (vi) too. Let $(\mathsf{w}', \mathsf{w}, \gamma')$ with $\mathsf{w}' = (\mathcal{E}_{\mathsf{w}'}, (r'', b))$ and $\mathsf{w} = (\mathcal{E}_{\mathsf{w}}, (r''_{[y \leftarrow \gamma(y)]}, b))$ be any triple added to Temp from $(\mathsf{v}', \mathsf{v}, \gamma)$.

(v) Let us first prove that $\mathcal{E}_{\mathsf{w}} \subseteq \{(\ell, C'_{[y \leftarrow \gamma'(y)]}, \top) \mid (\ell, C', \top) \in \mathcal{E}_{\mathsf{w}'}\} \cup \{(\ell, C, \bot)\}$. For any $(\ell, C, \top) \in \mathcal{E}_{\mathsf{w}}$, there exists $(\ell_0, C_0, \top) \in \mathcal{E}_{\mathsf{v}}$ such that by definition of the game:

- $\exists \ell_0 \xrightarrow{g,a,X'} \ell \in E$ s. t. $[r'_{[y \leftarrow \gamma(y)]} \cap C_0]_{|X} \cap g \neq \emptyset$
- $C = \overleftrightarrow{(r'_{[y \leftarrow \gamma(y)]} \cap C_0 \cap g)}_{[X' \leftarrow 0][\sigma(\mathsf{v}) \leftarrow 0]}$
- $[r'_{[y \leftarrow \gamma(y)]} \cap C_0]_{|X} \subseteq g$

By induction hypothesis, there exists $(\ell_0, C'_0, \top) \in \mathcal{E}_{v'}$ such that $C'_{0[y \leftarrow \gamma(y)]} = C_0$, then:

- $[r'_{[y \leftarrow \gamma(y)]} \cap C'_{0[y \leftarrow \gamma(y)]}]_{|X} \cap g \neq \emptyset$
- $[r'_{[y \leftarrow \gamma(y)]} \cap C'_{0[y \leftarrow \gamma(y)]}]_{|X} \subseteq g$

This implies that:

- $[r' \cap C'_0]_{|X} \cap g \neq \emptyset$
- $[r' \cap C'_0]_{|X} \subseteq g$

Indeed, $[r' \cap C'_0]_{|X} \subseteq [r'_{|Im\gamma \cup X} \cap C'_{0|Im\gamma\cup X}]_{|X} = [r'_{[y\leftarrow\gamma(y)]} \cap C'_{0[y\leftarrow\gamma(y)]}]_{|X}$. Then $[r'' \cap C'_0]_{|X} \cap g = \emptyset$ implies that $[r' \cap C'_0]_{|X} = \emptyset$ which is not possible if v' is a state of the game. As a consequence, there is a configuration $(\ell, C', \top)$ in $\mathcal{E}_{\mathsf{w}'}$ such that $C' = \overleftrightarrow{(r' \cap C'_0 \cap g)}_{[X'\leftarrow 0][\sigma'(\mathsf{v}')\leftarrow 0]}$. Then:

$$\begin{aligned}
C'_{[y\leftarrow\gamma'(y)]} &= \overleftrightarrow{(r' \cap C'_0 \cap g)}_{[X'\leftarrow 0][\sigma'(\mathsf{v}')\leftarrow 0][y\leftarrow\gamma'(y)]}\\
&= \overleftrightarrow{(r' \cap C'_0 \cap g)}_{[X'\leftarrow 0][\sigma'(\mathsf{v}')\leftarrow 0][y\leftarrow\gamma(y)][\sigma(\mathsf{v})\leftarrow\sigma'(\mathsf{v}')]}\\
&= \overleftrightarrow{(r'_{[y\leftarrow\gamma(y)]} \cap C'_{0[y\leftarrow\gamma(y)]} \cap g)}_{[X'\leftarrow 0][\sigma(\mathsf{v})\leftarrow\sigma'(\mathsf{v}')]}\\
&= \overleftrightarrow{(r'_{[y\leftarrow\gamma(y)]} \cap C'_{0[y\leftarrow\gamma(y)]} \cap g)}_{[X'\leftarrow 0][\sigma(\mathsf{v})\leftarrow 0]}.
\end{aligned}$$

Therefore, $(\mathsf{w}', \mathsf{w}, \gamma')$ satisfies (v). Finally, all the triples in $\mathsf{V_{def}}$ are added in Temp first, so that by induction v) is satisfied by all the triples in $\mathsf{V_{def}}$.

(vi) Let us now prove that $\mathcal{E}_{\mathsf{w}'} \subseteq \{(\ell, C', b') \mid (\ell, C'_{[y\leftarrow\gamma(y)]}, b) \in \mathcal{E}_{\mathsf{w}}\}$. For any $(\ell, C', b') \in \mathcal{E}_{\mathsf{w}'}$, there exists $(\ell_0, C'_0, b) \in \mathcal{E}_{\mathsf{v}'}$ such that:

- $\exists \ell_0 \xrightarrow{g,a,X'} \ell \in E$ s. t. $[r' \cap C'_0]_{|X} \cap g \neq \emptyset$
- $C = \overleftrightarrow{(r' \cap C_0 \cap g)}_{[X'\leftarrow 0][\sigma'(\mathsf{v}')\leftarrow 0]}$

By induction hypothesis, there exists $(\ell_0, C'_{0[y\leftarrow\gamma(y)]}, b) \in \mathcal{E}_{\mathsf{v}}$, then:

- $\emptyset \neq [r' \cap C'_0]_{|X} \cap g \subseteq [r'_{[y\leftarrow\gamma(y)]} \cap C'_{0[y\leftarrow\gamma(y)]}]_{|X} \cap g$

As a consequence, there is a configuration (ℓ, C, b) in $\mathcal{E}_{\mathsf{w}}$ such that the relation $C = \overleftrightarrow{(r'_{[y\leftarrow\gamma(y)]} \cap C'_{0[y\leftarrow\gamma(y)]} \cap g)}_{[X'\leftarrow 0][\sigma(\mathsf{v})\leftarrow 0]}$, which has been proved to be equal to $C'_{[y\leftarrow\gamma'(y)]}$ above. Therefore vi) is satisfied by $(\mathsf{w}', \mathsf{w}, \gamma')$ and, by induction, by all the triples in $\mathsf{V_{def}}$.

Finally, let us prove that (iv) is also satisfied for all triples in $\mathsf{V_{def}}$. Let $(\mathsf{v}', \mathsf{v}, \gamma)$ be a triple in $\mathsf{V_{def}}$. Let $\mathsf{w}' = (\mathcal{E}_{\mathsf{w}'}, (r'', b))$ be a state of Determinizator such that $\exists(\mathsf{v}'_S, r'', b) \in \mathsf{V}_S \times \mathsf{Act}_S$ such that $(\mathsf{v}', \sigma'(\mathsf{v}'), \mathsf{v}'_S) \in \delta \ \wedge\ (\mathsf{v}'_S, (r'', b), \mathsf{w}') \in \delta$. Then, there exists $(\ell_0, C'_0, b'_0) \in \mathcal{E}_{\mathsf{w}'}$ such that:

- $\exists \ell_0 \xrightarrow{g,b,X'} \ell' \in E$ s. t. $[r'' \cap C'_0]_{|X} \cap g \neq \emptyset$

Moreover, there exists $(\ell_1, C'_1, b'_1) \in \mathcal{E}_{\mathsf{v}'}$ such that:

- $\exists \ell_1 \xrightarrow{g_1,a,X'_1} \ell_0 \in E$ s. t. $[r' \cap C'_1]_{|X} \cap g_1 \neq \emptyset$
- $C'_0 = \overleftrightarrow{(r' \cap C'_1 \cap g_1)}_{[X'\leftarrow 0][\sigma'(\mathsf{v}')\leftarrow 0]}$.

As a consequence there exists a configuration $(\ell, C'_{1[y\leftarrow\gamma(y)]}, b_1) \in \mathcal{E}_v$ such that:

- $\emptyset \neq [r' \cap C'_1]_{|X} \cap g_1 \subseteq [r'_{[y\leftarrow\gamma(y)]} \cap C'_{1[y\leftarrow\gamma(y)]}]_{|X} \cap g_1$.

Then there exists a configuration (ℓ_0, C_0, b_0) of the successor v_S of v by the reset $\sigma(\mathsf{v})$ such that:

- $C_0 = \overleftrightarrow{(r'_{[y \leftarrow \gamma(y)]} \cap C'_{1[y \leftarrow \gamma(y)]} \cap g_1)}_{[X' \leftarrow 0][\sigma(\mathsf{v}) \leftarrow 0]}$

If we define γ' as expected ($\gamma'(y) = \gamma(y)$ if $y \notin \sigma(\mathsf{v})$, $\gamma'(y) = \sigma'(\mathsf{v}')$ otherwise) then $C_0 = C'_{0[y \leftarrow \gamma'(y)]}$. Moreover, $\emptyset \neq [r'' \cap C'_0]_{|X} \cap g \subseteq [r''_{[y \leftarrow \gamma'(y)]} \cap C'_{0[y \leftarrow \gamma'(y)]}]_{|X} \cap g$. Hence $(r''_{[y \leftarrow \gamma'(y)]}, b)$ is a possible move of Spoiler from v_S. Therefore there exists $\mathsf{w} \in \mathsf{V}_D$ such that $(\mathsf{v}, \sigma(\mathsf{v}), \mathsf{v}_S) \in \delta$ and $(\mathsf{v}_S, (r''_{[y \leftarrow \gamma'(y)]}, b), \mathsf{w}) \in \delta$. As a consequence, the triple $(\mathsf{w}', \mathsf{w}, \gamma')$ is added to Temp, then when this triple is taken from Temp, either $\sigma'(\mathsf{w}')$ is not already defined and this triple is added to $\mathsf{V}_{\mathsf{def}}$, or $\sigma'(\mathsf{w}')$ has been already defined and another triple of the form $(\mathsf{w}', \mathsf{w}_0, \gamma_0)$ has been added in $\mathsf{V}_{\mathsf{def}}$. Therefore (iv) is satisfied by all triples $(\mathsf{v}', \mathsf{v}, \gamma)$ in $\mathsf{V}_{\mathsf{def}}$. □

References

1. Alur R, Dill DL (1994) A theory of timed automata. Theor Comput Sci 126(2):183–235
2. Finkel O (2006) Undecidable problems about timed automata. In: Proceedings of the 4th international conference on formal modeling and analysis of timed systems (FORMATS'06) (Lecture Notes in Computer Science), vol 4202. Springer, pp. 187–199
3. Tripakis S (2006) Folk theorems on the determinization and minimization of timed automata. Inf Process Lett 99(6):222–226
4. Asarin E, Maler O, Pnueli A, Sifakis J (1998) Controller synthesis for timed automata. In: Proceedings of the 5th IFAC symposium on system structure and control (SSSC'98). Elsevier Science, Amsterdam, pp. 469–474
5. Alur R, Fix L, Henzinger TA (1994) A determinizable class of timed automata. In: Proceedings of the 6th international conference on computer aided verification (CAV'94) (Lecture Notes in Computer Science), vol 818. Springer, New York, pp 1–13
6. Vijay SP, Pandya PK, Narayanan KS, Lakshmi M (2008) 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) (Lecture Notes in Computer Science), vol 5215. Springer, pp. 78–92
7. Baier C, Bertrand N, Bouyer P, Brihaye T (2009) When are timed automata determinizable? In: Proceedings of the 36th international colloquium on automata, languages and programming (ICALP'09) (Lecture Notes in Computer Science), vol 5556. Springer, Rhodes, pp. 43–54
8. Krichen M, Tripakis S (2009) Conformance testing for real-time systems. Form Methods Syst Des 34(3):238–304
9. Bouyer P, Chevalier F, D'Souza D (2005) Fault diagnosis using timed automata. In: Proceedings of the 8th international conference on foundations of software science and computational structures (FOSSACS'05) (Lecture Notes in Computer Science), vol 3441. Springer, San Diego, pp. 219–233
10. Bertrand N, Jéron T, Stainer A, Krichen M (2012) Off-line test selection with test purposes for non-deterministic timed automata. Log Methods Comput Sci 8(4:8):1
11. Bertrand N, Stainer A, Jéron T, Krichen M (2011) A game approach to determinize timed automata. In: Proceedings of the 14th international conference on foundations of software science and computation structures (FOSSACS'11) (Lecture Notes in Computer Science), vol 6604. Springer, pp. 245–259.
12. Grädel E, Thomas W, Wilke T (2002) (eds) Automata, logics, and infinite games: a guide to current research (Lecture Notes in Computer Science), (vol 2500). Springer, Berlin
13. Bouyer P, Dufourd C, Fleury E, Petit A (2004) Updatable timed automata. Theor Comput Sci 321(2–3):291–345
14. Lakshmi M, Narayanan KS (2010) Integer reset timed automata: clock reduction and determinizability. CoRR arXiv:1001.1215v1
15. Bouyer P (2004) Forward analysis of updatable timed automata. Form Methods Syst Des 24(3):281–320
16. Bertrand N, Jéron T, Stainer A, Krichen M (2011) Off-line test selection with test purposes for non-deterministic timed automata. In: Proceedings of the 17th international conference on tools and algorithms for the construction and analysis of systems (TACAS'11) (Lecture Notes in Computer Science), vol 6605. Springer, pp. 96–111

17. Alur R, Henzinger TA, Kupferman O, Vardi MY (1998) Alternating refinement relations. In: Proceedings of the 9th international conference on concurrency theory (CONCUR'98) (Lecture Notes in Computer Science), vol 1466. Springer, New York, pp 163–178
18. David A, Larsen KG, Legay A, Nyman U, Wasowski A (2010) 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). ACM, pp. 91–100