# When Are Timed Automata Determinizable?

Christel Baier[1], Nathalie Bertrand[2], Patricia Bouyer[3], and Thomas Brihaye[4]

[1] Technische Universität Dresden, Germany
[2] INRIA Rennes Bretagne Atlantique, France
[3] LSV, CNRS & ENS Cachan, France
[4] Université de Mons, Belgium

**Abstract.** In this paper, we propose an abstract procedure which, given a timed automaton, produces a language-equivalent deterministic infinite timed tree. We prove that under a certain boundedness condition, the infinite timed tree can be reduced into a classical *deterministic* timed automaton. The boundedness condition is satisfied by several subclasses of timed automata, some of them were known to be determinizable (event-clock timed automata, automata with integer resets), but some others were not. We prove for instance that strongly non-Zeno timed automata can be determinized. As a corollary of those constructions, we get for those classes the decidability of the universality and of the inclusion problems, and compute their complexities (the inclusion problem is for instance EXPSPACE-complete for strongly non-Zeno timed automata).

## 1 Introduction

Timed automata have been proposed by Alur and Dill in the early 90s as a model for real-time systems [2]. A timed automaton is a finite automaton which can manipulate real-valued variables called clocks, that evolve synchronously with the time, can be tested and reset to zero. One of the fundamental properties of this model is that, although the set of configurations is in general infinite, checking reachability properties is decidable. From a language-theoretic point of view, this means that checking emptiness of the timed language accepted by a timed automaton can be decided (and is a PSPACE-complete problem). The proof relies on the construction of the so-called region automaton, which finitely abstracts behaviours of a timed automaton. Since then, its appropriateness as a model for the verification of real-time systems has been confirmed, with the development of verification algorithms and dedicated tools.

There are however two weaknesses to that model: a timed automaton cannot be determinized, and inclusion (and universality) checking is undecidable [2], except for deterministic timed automata. This basically forbids the use of timed automata as a specification language. Understanding and coping with these weaknesses have attracted lots of research, and, for instance, testing whether a timed automaton is determinizable has been proved undecidable [6]. Also, the undecidability of universality has been further investigated, and rather restricted classes of timed automata suffer from that undecidability result [1]. On the other hand, classes of timed automata have been exhibited, that either can be effectively determinized (for instance event-clock timed automata [3], or timed automata with integer resets [9]), or for which universality can be decided (for instance single-clock timed automata [7]).

In this paper, we describe a generic construction that is applicable to every timed automaton, and which, under certain conditions, yields a deterministic timed automaton, which is language-equivalent to the original timed automaton. The idea of the procedure is to unfold the timed automaton into a finitely-branching infinite tree that records the timing constraints that have to be satisfied using one clock per level of the tree (hence infinitely many clocks). When reading a finite timed word in that infinite tree, we may reach several nodes of the tree, but the timing information stored in the clocks is independent of the run in the tree. Thanks to this kind of *input-determinacy* property, we can determinize this infinite object, yielding another finitely-branching infinite tree. And, under a boundedness condition on the amount of timing information we need to store, we will be able to fold back the tree into a deterministic timed automaton. This boundedness condition is not a syntactical condition on the original timed automaton, but will be satisfied by large classes of timed automata: event-clock timed automata [3], timed automata with integer resets [9], and strongly non-Zeno timed automata. Furthermore, our construction yields automata of exponential-size in the first case, and doubly-exponential-size automata otherwise. In particular, our approach provides an EXPSPACE algorithm to check universality (and inclusion) for a large class of timed automata, and we prove that this complexity is tight. Our algorithm can easily be adapted into a PSPACE one, in the special case of event-clock timed automata, allowing to recover the known result of [3].

## 2 Timed Automata

**Preliminaries.** Given $X$ a finite or infinite set of clocks and $M$ a non-negative integer, a clock *valuation* over $X$ bounded by $M$ is a mapping $v : X \to \mathbb{T}_M$ where $\mathbb{T}_M = [0, M] \cup \{\bot\}$. We assume furthermore that $\bot > M$. The notation $\bot$ is for abstracting values of clocks that are above some fixed value $M$. This is rather non-standard (though used for instance in [8]) but it will be convenient in this paper. We note $\bar{0}$ the valuation that assigns 0 to all clocks. If $v$ is a valuation over $X$ and bounded by $M$, and $t \in \mathbb{R}_+$, then $v + t$ denotes the valuation which assigns to every clock $x \in X$ the value $v(x) + t$ if $v(x) + t \leq M$, and $\bot$ otherwise (in particular, if $v(x) = \bot$, then $(v + t)(x) = \bot$). For $Y \subseteq X$ we write $[Y \leftarrow 0]v$ for the valuation equal to $v$ on $X \setminus Y$ and to $\bar{0}$ on $Y$, and $v_{|Y}$ for the valuation $v$ restricted to clocks in $Y$. A(n $M$-bounded) *guard* (or *constraint*) over $X$ is a finite conjunction of constraints of the form $x \sim c$ where $x \in X$, $c \in \mathbb{N} \cap [0, M]$ and $\sim \in \{<, \leq, =, \geq, >\}$. We denote by $\mathcal{G}_M(X)$ the set of $M$-bounded guards over $X$. Given a valuation $v$ and a guard $g$ we write $v \models g$ whenever $v$ satisfies $g$.

A timed word over $\Sigma$ is a finite sequence of pairs $(a_1, t_1)(a_2, t_2) \ldots (a_k, t_k)$ such that for every $i$, $a_i \in \Sigma$ and $(t_i)_{1 \leq i \leq k}$ is a nondecreasing sequence in $\mathbb{R}_+$.

**Timed automata.** A *timed automaton* is a tuple $\mathcal{A} = (L, \ell_0, L_{\mathsf{acc}}, X, M, E)$ such that: $(i)$ $L$ is a finite set of locations, $(ii)$ $\ell_0 \in L$ is the initial location, $(iii)$ $L_{\mathsf{acc}} \subseteq L$ is the set of final locations, $(iv)$ $X$ is a finite set of clocks, $(v)$ $M \in \mathbb{N}$, and $(vi)$ $E \subseteq L \times \mathcal{G}_M(X) \times \Sigma \times 2^X \times L$ is a finite set of edges. Constant $M$ is called the maximal constant of $\mathcal{A}$.

The semantics of a timed automaton $\mathcal{A}$ is given as a timed transition system $\mathcal{T}_{\mathcal{A}} = (S, s_0, S_{\mathsf{acc}}, (\mathbb{R}_+ \times \Sigma), \rightarrow)$ with set of states $S = L \times \mathbb{T}_M^X$, initial state $s_0 = (\ell_0, \bar{0})$, set of accepting states $S_{\mathsf{acc}} = L_{\mathsf{acc}} \times \mathbb{T}_M^X$, and transition relation $\rightarrow \subseteq S \times (\mathbb{R}_+ \times \Sigma) \times S$ composed of moves of the form $(\ell, v) \xrightarrow{\tau,a} (\ell', v')$ whenever there exists an edge $(\ell, g, a, Y, \ell') \in E$ such that $v+\tau \models g$ and $v' = [Y \leftarrow 0](v+\tau)$.

A run $\varrho$ of $\mathcal{A}$ is a finite sequence of moves, *i.e.*, $\varrho = s_0 \xrightarrow{\tau_1,a_1} s_1 \dots \xrightarrow{\tau_k,a_k} s_k$. It is said initial whenever $s_0 = (\ell_0, \bar{0})$. An initial run is accepting if it ends in an accepting location. The timed word $u = (a_1, t_1)(a_2, t_2) \dots (a_k, t_k)$ is said to be read on $\varrho$ whenever $t_i = \sum_{j=1}^{i} \tau_j$ for every $1 \leq i \leq k$. We write $\mathcal{L}(\mathcal{A})$ for the set of timed words (or timed language) accepted by $\mathcal{A}$, that is the set of timed words $u$ such that there exists an initial and accepting run $\varrho$ which reads $u$.

A timed automaton $\mathcal{A}$ is *deterministic* whenever for every timed word $u$, there is at most one initial run which reads $u$. It is *strongly non-Zeno* whenever there exists $K \in \mathbb{N}$ such that for every run $\varrho = s_0 \xrightarrow{\tau_1,a_1} s_1 \dots \xrightarrow{\tau_k,a_k} s_k$ in $\mathcal{A}$, $k \geq K$ implies $\sum_{i=1}^{k} \tau_i \geq 1$. This condition is rather standard in timed automata [4].

*Example 1.* An example of timed automaton is depicted in Fig. 1. This automaton will be used as a running example throughout the paper in order to illustrate the different steps of our construction. This automaton is not deterministic and accepts the timed language $\{(a, t_1)(a, t_2) \cdots (a, t_{2n}) \mid n \geq 1,\ 0 < t_1 < t_2 < \cdots < t_{2n-1}$ and $t_{2n} - t_{2n-2} = 1\}$, with the convention that $t_0 = 0$. The timed word $(a, 0.5)(a, 1.6)(a, 2.9)$ can be read on the initial run $(\ell_0, 0) \xrightarrow{0.5,a} (\ell_3, 0) \xrightarrow{1.1,a} (\ell_0, 0) \xrightarrow{1.3,a} (\ell_1, \perp)$ but is not accepted. The last configuration of the above run is $(\ell_1, \perp)$ because the value of clock $x$ should be 1.3, but as it is larger than the maximal constant 1, we abstract the precise value into $\perp$.

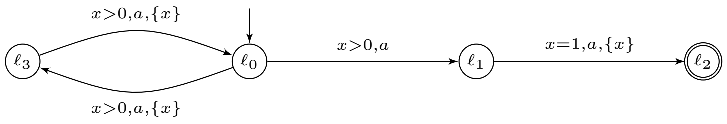


**Fig. 1.** A timed automaton $\mathcal{A}$

**On timed bisimulations.** A *strong timed (resp. time-abstract) simulation relation* between two timed transition systems $\mathcal{T}_i = (S_i, s_{i,0}, S_{i,\mathsf{acc}}, (\Sigma \cup \mathbb{R}_+), \rightarrow_i)$ for $i \in \{1, 2\}$ is a relation $\mathfrak{R} \subseteq S_1 \times S_2$ such that if $s_1 \mathrel{\mathfrak{R}} s_2$ and $s_1 \xrightarrow{t_1,a} s_1'$ for some $t_1 \in \mathbb{R}_+$ and $a \in \Sigma$, then there exists $s_2' \in S_2$ (resp. $t_2 \in \mathbb{R}_+$ and $s_2' \in S_2$) such that $s_2 \xrightarrow{t_1,a} s_2'$ (resp. $s_2 \xrightarrow{t_2,a} s_2'$) and $s_1' \mathrel{\mathfrak{R}} s_2'$. A *strong timed (resp. time-abstract) bisimulation relation* between two timed transition $\mathcal{T}_i$ for $i \in \{1, 2\}$ is a relation $\mathfrak{R} \subseteq S_1 \times S_2$ such that both $\mathfrak{R}$ and $\mathfrak{R}^{-1}$ are strong timed (resp. time-abstract) simulation relations. The above relations *preserve* initial (resp.

accepting) states whenever $s_{1,0} \mathfrak{R} s_{2,0}$ (resp. $s_1 \mathfrak{R} s_2$ and $s_i \in S_{i,\mathsf{acc}}$ implies $s_{3-i} \in S_{3-i,\mathsf{acc}}$). Note that the notion of strong timed bisimulation which preserves initial and accepting states is stronger than that of language equivalence.

**The classical region construction.** We let $X$ be a finite set of clocks, and $M \in \mathbb{N}$. We define the equivalence relation $\equiv_{X,M}$ between valuations in $\mathbb{T}_M$ as follows: $v \equiv_{X,M} v'$ iff $(i)$ for every clock $x \in X$, $v(x) \le M$ iff $v'(x) \le M$; $(ii)$ for every clock $x \in X$, if $v(x) \le M$, then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, and $(ii)$ for every pair of clocks $(x, y) \in X^2$ such that $v(x) \le M$ and $v'(x) \le M$, $\{v(x)\} \le \{v(y)\}$ iff $\{v'(x)\} \le \{v'(y)\}$. [1] 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*. We note $\mathsf{Reg}_M^X$ for the set of such regions. A region $r'$ is a time-successor of a region $r$ if there is $v \in r$ and $t \in \mathbb{R}_+$ such that $v + t \in r'$. If $v$ is a valuation, we will write $[v]$ for the region to which $v$ belongs (when $X$ and $M$ are clear in the context).

It is a classical result [2] that given a timed automaton $\mathcal{A}$ with maximal constant $M$ and set of clocks $X$, the relation $\mathfrak{R}_{X,M}$ between configurations of $\mathcal{A}$ defined by $(\ell, v) \mathfrak{R}_{X,M} (\ell, v')$ iff $v \equiv_{X,M} v'$ is a time-abstract bisimulation.

## 3 Some Transformations

In this section, we describe a general construction that aims at determinizing a timed automaton. We know however that not all timed automata can be determinized [2], and even that we cannot decide whether a timed automaton can be determinized [6]. We will thus give conditions that will ensure $(i)$ that our procedure can be properly applied, and $(ii)$ that the resulting timed automaton is deterministic and accepts the same timed language as the original automaton. We will then analyze the complexity of the procedure, and apply it to several subclasses of timed automata, some of which were known to be determinizable, some other were not known to be determinizable.

This construction consists in four steps: $(i)$ an unfolding of the original automaton into an infinite timed tree, $(ii)$ a region abstraction, $(iii)$ a symbolic determinization, and $(iv)$ a reduction of the number of clocks, allowing to fold the tree back into a timed automaton. These steps are described in the following subsections. Due to page limitation, we will give no formal definitions of the objects we build in our construction, and better illustrate the construction on the running example. All details can be found in the technical report [5].

### 3.1 Construction of an Equivalent Infinite Timed Tree

In this first step, we unfold the timed automaton $\mathcal{A}$ into a finitely-branching *infinite timed tree* $\mathcal{A}^\infty$ that has infinitely many clocks (one clock per level of the tree), we call $Z = \{z_0, z_1, \ldots\}$ this infinite set of clocks. The idea of this unfolding is to use a fresh clock reset at each level of the tree in order to record the timing constraints that have to be satisfied in $\mathcal{A}$. Each node $n$ of $\mathcal{A}^\infty$ is

[1] Where $\lfloor \alpha \rfloor$ (resp. $\{\alpha\}$) denotes the integral (resp. fractional) part of $\alpha$.

labelled by a pair $(\ell, \sigma) \in L \times Z^X$ where $\ell$ records the location of $\mathcal{A}$ that node $n$ simulates and $\sigma$ describes how the clocks of $\mathcal{A}$ are encoded using the clocks of $\mathcal{A}^\infty$ (if $\sigma(x) = z_i$, the value clock $x$ would have in $\mathcal{A}$ is the current value of clock $z_i$). The advantage of this infinite timed tree is that it enjoys some *input-determinacy* property: when reading a finite timed word $u$ in $\mathcal{A}^\infty$, there may be several runs in the tree that read $u$, but the timing information stored in the clocks is independent of the run in the tree (see Remark 4).

clock $x_i$ is the time elapsed since letter $a_i$ is read.

*Example 2.* Part of the infinite timed tree $\mathcal{A}^\infty$ associated with the timed automaton $\mathcal{A}$ of Fig. 1 is depicted in Fig. 2. Notice that a fresh clock is reset at each level; for instance $z_2$ is reset on all edges from level-1 to level-2 nodes (*i.e.* $n_1 \to n_3$ and $n_2 \to n_4$). The timed tree $\mathcal{A}^\infty$ corresponds to the unfolding of $\mathcal{A}$: the two branches starting from the node $n_0$ represent the possible choice in state $\ell_0$ of $\mathcal{A}$; the same phenomenon also happens in $n_4$. The label of $n_4$ is $(\ell_0, z_2)$; it means that node $n_4$ represents the location $\ell_0$ of $\mathcal{A}$ and that the value of clock $x$ can be recovered from the current value of clock $z_2$. It is important to observe how the second component of the label evolves. First consider the edge $n_4 \to n_5$; it represents the transition from $\ell_0$ to $\ell_1$ in $\mathcal{A}$, which does not reset clock $x$; the reference for clock $x$ is the same in $n_5$ as it is in $n_4$, that is why the label of $n_5$ is $(\ell_1, z_2)$. Now consider the edge $n_4 \to n_6$; it represents the transition from $\ell_0$ to $\ell_3$ in $\mathcal{A}$, which resets clock $x$; the reference for clock $x$ thus becomes $z_3$, the clock which has just been reset, that is why the label of $n_6$ is $(\ell_3, z_3)$.

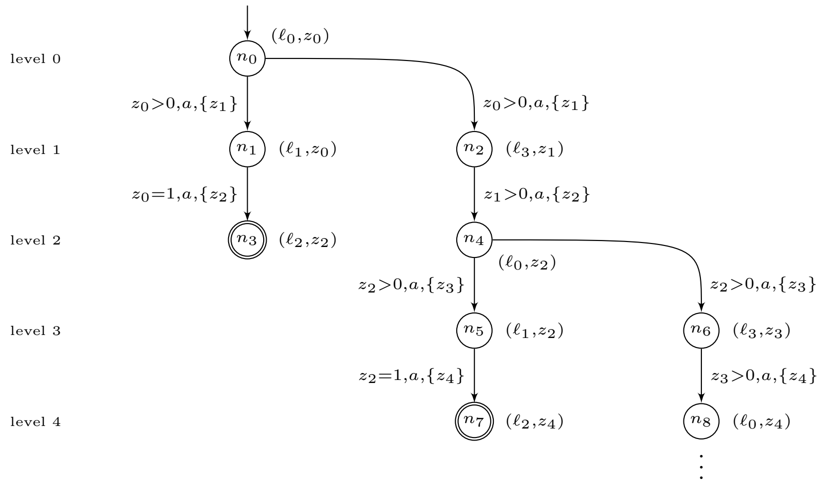


**Fig. 2.** The infinite timed tree $\mathcal{A}^\infty$ associated with the timed automaton $\mathcal{A}$ of Fig. 1

The correctness of this unfolding is stated in the follow lemma.

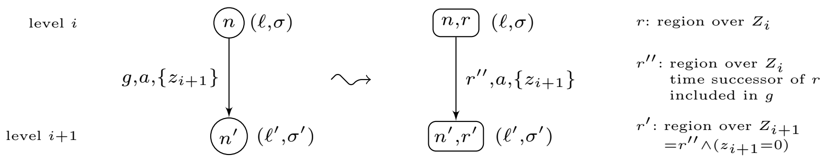
**Lemma 3.** *The relation* $\mathfrak{R}_1$ *between states of* $\mathcal{A}$ *and states of* $\mathcal{A}^\infty$ *defined by* $(\ell, v \circ \sigma)\ \mathfrak{R}_1\ (n, v)$ *if* $label(n) = (\ell, \sigma)$ *is a strong timed bisimulation.*

$A \approx A^\infty$

*Remark 4.* In $\mathcal{A}^\infty$, for every finite timed word $u$, there is a unique valuation $v_u \in \mathbb{T}^Z$ such that for every initial run $\varrho$ in $\mathcal{A}^\infty$ that reads $u$, $\varrho$ ends in some configuration $(n, v_u)$ with $level(n) = |u|$. Indeed, if the timed word $u$ is of the form $(a_1, t_1)...(a_{|u|}, t_{|u|})$, any initial run $\varrho$ reading $u$ necessarily ends in a configuration $(n, v_u)$ where $level(n) = |u|$ and $v_u(z_j) = t_{|u|} - t_j$ for any $j \leq |u|$.

## 3.2 A Region Abstraction

In this second step, we extend in a natural way the classical region equivalence to the above infinite timed tree: at level $i$ of the tree, only clocks in $Z_i = \{z_0, \cdots, z_i\}$ are relevant (all other clocks have not been used yet), we thus consider regions over that set of clocks. We use $R(\mathcal{A}^\infty)$ to denote this region abstraction, and we interpret it in a timed manner. We do not illustrate this transformation step on our running example, since $R(\mathcal{A}^\infty)$ is easily obtained from $\mathcal{A}^\infty$, but only depict the transformation on an edge, see below:



It is worth noting that, in $R(\mathcal{A}^\infty)$, any state reached after a transition is of the form $((n, r), v)$, where $n$ is a node of $\mathcal{A}^\infty$ (of some level, say $i$), $r$ is a region over $Z_i$, and $v$ is a valuation over $Z_i$ which belongs to $r$. It is not difficult to see that, as in the standard region construction in timed automata, two states $((n, r), v_1)$ and $((n, r), v_2)$ with $v_1, v_2 \in r$ are time-abstract bisimilar. Furthermore, $R(\mathcal{A}^\infty)$ will satisfy the same input-determinacy property as $\mathcal{A}^\infty$ (see Remark 4). The correctness of $R(\mathcal{A}^\infty)$ can then be stated as follows.

**Lemma 5.** *The relation $\mathfrak{R}_2$ between states of $\mathcal{A}^\infty$ and states of $R(\mathcal{A}^\infty)$ defined by $(n, v)\ \mathfrak{R}_2\ ((n, r), v)$ if $v \in r$ is a strong timed bisimulation.*

$A^\infty \approx R(A^\infty)$

## 3.3 Symbolic Determinization

This third step is the crucial step of our construction. We will symbolically determinize the infinite timed tree $R(\mathcal{A}^\infty)$ using a rather standard subset construction, and we denote by $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ the resulting infinite tree. However there will be a subtlety in the subset construction: useless clocks will be forgotten 'on-the-fly'. More precisely, at each node, we only consider active clocks, *i.e.* clocks that appear in the label of the node (other clocks record values that do not impact on further behaviours of the system). The determinization is then performed on the 'symbolic' alphabet composed of regions over active clocks and actions, and thanks to the input-determinacy property of $R(\mathcal{A}^\infty)$, this symbolic determinization coincides with the determinization of the underlying timed transition system. Let us explain this crucial step on our running example.

projection

*Example 6.* The construction of $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is illustrated on Fig. 3. The determinization is performed using a classical subset construction. For example starting from node $n_0$, both $n_1$ and $n_2$ can be reached *via* a transition with guard $0 < z_0 < 1$. This is reflected in the leftmost $\{n_1, n_2\}$-node at the first level. It is also important to understand the meaning of active clocks. In $\mathcal{A}^\infty$, the only active clock in node $n_4$ is $z_2$. Therefore, guards on transitions leaving the node $(\{n_4\}, z_2 = 0)$ in $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ are regions over this unique clock $z_2$. If we consider a node combining $n_5$ and $n_6$, active clocks will consist in the union of active clocks in both nodes, hence $z_2$ and $z_3$. For sake of readability, we have mostly omitted labels of nodes on Fig. 3, but they can be naturally inferred from those in $R(\mathcal{A}^\infty)$; for instance, the label of the top-rightmost node is $\{(\ell_1, z_0), (\ell_3, z_1)\}$, the union of the labels of $n_1$ and $n_2$ in $R(\mathcal{A}^\infty)$.

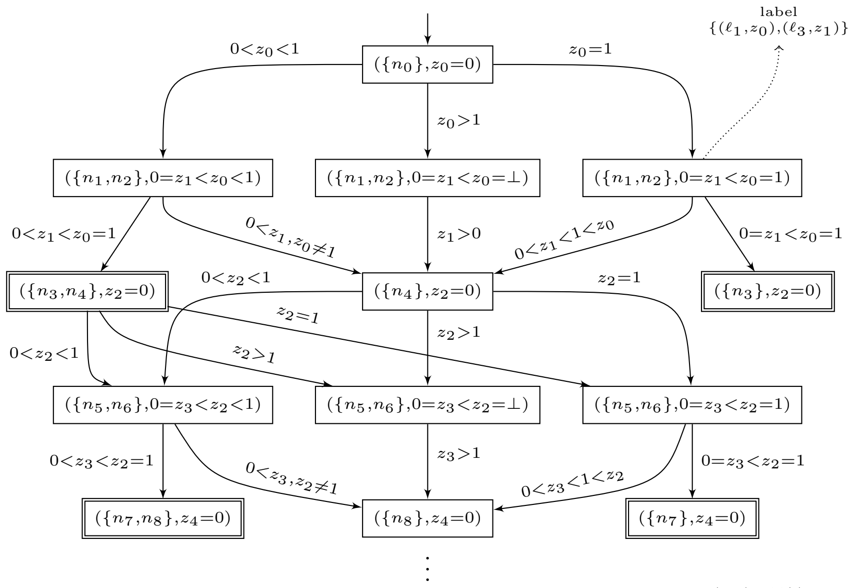


**Fig. 3.** The DAG induced by the infinite timed tree $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$

The subset construction induces a DAG (as seen on Fig. 3). However the rest of the construction will require a tree instead of a DAG; we thus add markers to nodes, so that we can have several copies of a node, depending on the ancestors. A node in $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is thus a tuple $(\star, K, r)$ where $\star$ is a marker, $K$ is a subset of node names in $R(\mathcal{A}^\infty)$ (they all have ~~same level~~), and $r$ is a region over the set $Act(K) = \bigcup_{n \in K, label(n)=(\ell,\sigma)} \sigma(X)$, the set of active clocks in $K$.

ACTIVE CLOCKS $\rightarrow$

The correctness of $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is stated in the following proposition.

**Proposition 7.** $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ *is a deterministic timed tree, and for every node* $N = (\star, K, r)$ *and for every valuation* $v \in \mathbb{T}^{Act(K)}$ *with* $v \in r$,

$$\mathcal{L}(\mathsf{SymbDet}(R(\mathcal{A}^\infty)), (N, v)) = \bigcup_{n \in K} \mathcal{L}(R(\mathcal{A}^\infty), ((n, r), v))$$

$Act(K) \subseteq Z_i$
if $K$ at level $i$

*Remark 8.* In case $\mathcal{A}$ has a single clock $x$, a level-$i$ node of $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ carries the following information: a finite set of pairs of the form $(\ell, x \mapsto z_j)$ for some $j \le i$ and a region for ACTIVE clocks in $Z_i$. We skip details, but with this information, we can easily recover the well-quasi-order that gives the decidability of the universality problem in single-clock timed automata [7].

## 3.4 Reduction of the Number of Clocks

$\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is an infinite object (it is an infinite timed tree and it has infinitely many clocks). Our aim is to fold this tree back into a deterministic timed automaton. Obviously we cannot do so for all timed automata, and so far we have not made any assumption on $\mathcal{A}$. Given $\gamma \in \mathbb{N}$, we say that $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is *$\gamma$-clock-bounded* if in every node, the number of active clocks is bounded by $\gamma$. Under this hypothesis, we will be able to quotient $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ by an equivalence of finite index, and get a deterministic timed automaton $\mathcal{B}_{\mathcal{A},\gamma}$ which accepts the same language as the original timed automaton $\mathcal{A}$.

The idea will be to fix a finite set of clocks $X_\gamma = \{x_1, \cdots, x_\gamma\}$, and starting from the level-0 node of $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ to rename the active clocks into clocks in $X_\gamma$ following a deterministic policy. Under the $\gamma$-clock-boundedness assumption, each time we will require a new clock (because a new one has become active), there will be (at least) one free clock in $X_\gamma$. Of course, we rename clocks in guards and regions as well, and change the labels of the nodes accordingly (an element of the label of a node is now a pair $(\ell, \sigma)$ where $\ell$ is a location of $\mathcal{A}$ and $\sigma \colon X \mapsto X_\gamma$ assigns to each clock of $\mathcal{A}$ its representative in the tree). The new object is still infinite, but it has finitely many clocks. A node is now a tuple $(\star, K, r)$ where $\star$ is a marker, $K$ is a subset of nodes in $R(\mathcal{A}^\infty)$ and $r$ is a region over (a subset of) $X_\gamma$. Now it is just a matter of noticing that two nodes with the same region and the same labels are isomorphic and strongly timed bisimilar (in particular they are language-equivalent). Timed automaton $\mathcal{B}_{\mathcal{A},\gamma}$ is obtained by merging such nodes.

*Example 9.* In Fig. 3, it is easy to see that $\mathsf{SymbDet}(\mathcal{A}^\infty)$ is 2-clock-bounded. So one can rename the clocks to $X_2 = \{x_1, x_2\}$, for instance we can map clocks with even indices to $x_1$ and clocks with odd indices to $x_2$. After this renaming, nodes sharing the same label (that is: set of locations of $\mathcal{A}$, mappings from $X$ to $\{x_1, x_2\}$ and regions over $\{x_1, x_2\}$) can be merged. Indeed, one can show that subtrees rooted at nodes with the same label are strongly timed bisimilar. For instance, in our running example, nodes $(\{n_0\}, z_0 = 0)$ and $(\{n_4\}, z_2 = 0)$, labelled respectively by $\{(\ell_0, z_0)\}$ and $\{(\ell_0, z_2)\}$ in $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$, are merged into a single location with region $x_1 = 0$. The resulting timed automaton is depicted on Fig. 4. In general, a location of this automaton is of the form $(\{(\ell_j, \sigma_j) \mid j \in J\}, r)$ where $J$ is a finite set, $\ell_j$ is a location of $\mathcal{A}$, $\sigma_j \colon X \to X_2$, and $r$ is a region over a subset of $X_2$. In our running example, there is a single clock $x$, hence we assimilate $\sigma_j$ with the value $\sigma_j(x)$.

The correctness of the construction is stated in the following proposition.

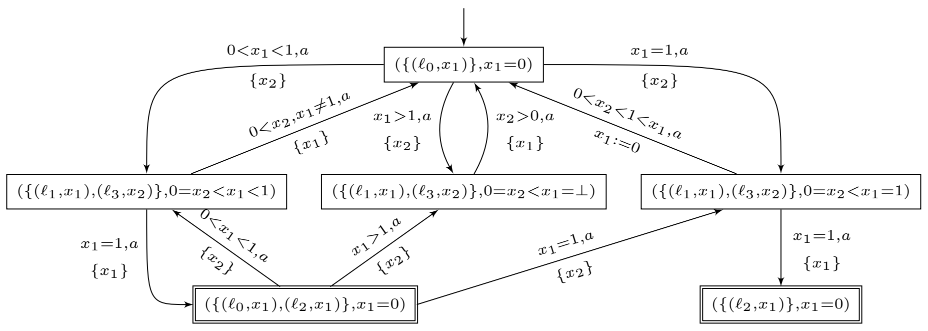


**Fig. 4.** The deterministic version of $\mathcal{A}$: the timed automaton $\mathcal{B}_{\mathcal{A},\gamma}$

**Proposition 10.** *Assume that* $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ *is* $\gamma$*-clock-bounded. Then,* $\mathcal{B}_{\mathcal{A},\gamma}$ *is a deterministic timed automaton, and* $\mathcal{L}(\mathcal{B}_{\mathcal{A},\gamma}) = \mathcal{L}(\mathcal{A})$.

### 3.5 Algorithmic Issues and Complexity

In this subsection, we shortly discuss the size of the effectiveness of its construction. If $\mathcal{A} = (L, \ell_0, L_{\mathsf{acc}}, X, M, E)$ is a timed automaton such that $\mathsf{SymbDet}$ $(R(\mathcal{A}^\infty))$ is $\gamma$-clock-bounded (for some $\gamma \in \mathbb{N}$), then the timed automaton $\mathcal{B}_{\mathcal{A},\gamma}$ has roughly $\underbrace{\alpha(\mathcal{A},\gamma) = 2^{|L|} \cdot \underline{\gamma}^{|X|} \cdot \Big((2M+2)^{(\underline{\gamma+1})^2} \cdot \underline{\gamma}!\Big) \text{ locations}}$ because a location is characterized by a finite set of pairs $(\ell, \sigma)$ with $\ell$ a location of $\mathcal{A}$, $\sigma\colon X \to X_\gamma$, and a region over $X_\gamma$.

The procedure we have described goes through the construction of infinite objects. However, if we abstract away the complete construction, we know precisely how locations and transitions are derived. Hence, $\mathcal{B}_{\mathcal{A},\gamma}$ can be computed on-the-fly by guessing new transitions, and so can its complement (since $\mathcal{B}_{\mathcal{A},\gamma}$ is deterministic). A location of the automaton $\mathcal{B}_{\mathcal{A},\gamma}$ can be stored in space logarithmic in $\alpha(\mathcal{A},\gamma)$, and we will thus be able to check for universality (e.g.) in nondeterministic space $\log(\alpha(\mathcal{A},\gamma))$.

## 4 Our Results

We will now investigate several classes of timed automata for which the procedure described in Section 3 applies.

### 4.1 Some Classes of Timed Automata Are Determinizable

**Automata satisfying the $p$-assumption ($\mathsf{TA}_p$).** Given $p \in \mathbb{N}$, we say that a timed automaton $\mathcal{A}$ satisfies the *$p$-assumption* if for every $n \geq p$, for every run $\varrho = (\ell_0, v_0) \xrightarrow{\tau_1, a_1} (\ell_1, v_1) \ldots \xrightarrow{\tau_n, a_n} (\ell_n, v_n)$ in $\mathcal{A}$, for every clock $x \in X$, either $x$ is reset along $\varrho$ or $v_n(x) = \bot$. This assumption will ensure that we can apply the previous procedure, because if $\mathcal{A}$ satisfies the $p$-assumption, $\mathsf{SymbDet}(R(\mathcal{A}^\infty))$ is

$p$-clock-bounded. Then we observe that any strongly non-Zeno timed automaton (we write $\mathsf{SnZTA}$ for this class) satisfies the $p$-assumption for some $p \in \mathbb{N}$ which is exponential in the size of the automaton. We thus get the following result:

**Theorem 11.** *For every timed automaton $\mathcal{A}$ in $\mathsf{SnZTA}$ or in $\mathsf{TA}_p$, we can construct a deterministic timed automaton $\mathcal{B}$, whose size is doubly-exponential in the size of $\mathcal{A}$, and which recognizes the same language as $\mathcal{A}$.*

**Event-clock timed automata ($\mathsf{ECTA}$) [3].** An *event-clock timed automaton* is a timed automaton that contains only event-recording clocks: for every letter $a \in \Sigma$, there is a clock $x_a$, which is reset at every occurrence of $a$. It is easy to see that the deterministic timed tree associated with such an automaton is $|\Sigma|$-clock-bounded. Thus, applying our procedure, we recover the result of [3], with the same complexity bound.

**Theorem 12.** *For every timed automaton $\mathcal{A}$ in $\mathsf{ECTA}$, we can construct a deterministic timed automaton $\mathcal{B}$, whose size is exponential in the size of $\mathcal{A}$, and which recognizes the same language as $\mathcal{A}$.*

**Timed automata with integer resets ($\mathsf{IRTA}$) [9].** A *timed automaton with integer resets* is a timed automaton in which every edge $e = (\ell, g, a, Y, \ell')$ is such that $Y$ is non empty if and only if $g$ contains at least one atomic constraint of the form $x = c$, for some clock $x$. In that case, we observe that the deterministic timed tree associated with such an automaton is $(M+1)$-clock-bounded. We thus recover the result of [9], with the same complexity bound.

**Theorem 13.** *For every timed automaton $\mathcal{A}$ in $\mathsf{IRTA}$, we can construct a deterministic timed automaton $\mathcal{B}$, whose size is doubly-exponential in the size of $\mathcal{A}$, and which recognizes the same language as $\mathcal{A}$.*

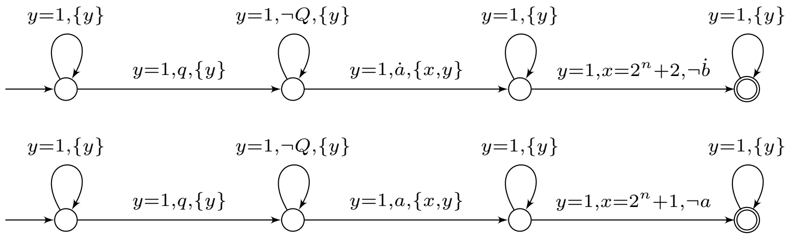
## 4.2 Deciding Universality and Inclusion

The universality and the inclusion problems are undecidable for the general class of timed automata [2]. Given $\mathcal{A}$ and $\mathcal{B}$ two timed automata, the *universality problem* asks whether $\mathcal{L}(\mathcal{A})$ is the set of all finite timed words, and the inclusion problem asks whether $\mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{A})$. When $\mathcal{A}$ belongs to one of the above determinizable classes, we will be able to decide the universality and the inclusion problems (there is no need to restrict automaton $\mathcal{B}$). We establish now the precise complexity of those problems, and start by providing a lower bound for the universality problem.

**Proposition 14.** *Checking universality in timed automata either satisfying the $p$-assumption for some $p$ or with integer resets is EXPSPACE-hard.*

*Proof (sketch).* The idea of the proof is as follows. Given an exponential-space Turing machine $\mathcal{M}$ with input word $w_0$, we define a timed automaton $\mathcal{A}_{\mathcal{M},w_0}$ such that $\mathcal{A}_{\mathcal{M},w_0}$ is universal if and only if $\mathcal{M}$ does not halt on input $w_0$. An execution of $\mathcal{M}$ over $w_0$ is encoded by a timed word, and $\mathcal{A}_{\mathcal{M},w_0}$ will accept all finite timed words that are not encodings of halting executions of $\mathcal{M}$ on $w_0$. Assuming $|w_0| = n$, the maximal length of the tape is $2^n$, and a configuration of $\mathcal{M}$ can be seen as a pair $\langle q, w\rangle$, where $q$ is a control state of $\mathcal{M}$ and $w$ is a word of length $2^n$ that represents the content of the tape (the position of the tape head is marked by a dotted letter). We furthermore require that actions are separated by precisely one time unit, which entails for instance that control states should be separated by precisely $2^n + 1$ time units.

A finite timed word might not be the encoding of an halting computation in $\mathcal{M}$ for several reasons: it is not the encoding of a proper execution in $\mathcal{M}$, or it does not end in the halting state, or actions do not occur at integer time points, or control states are not separated by $2^n+1$ time units, *etc.* All these properties can be described using either timed automata satisfying the $p$-assumption, or timed automata with integer resets. For instance, a rule of the form $(q, a, b, \mathsf{right}, q')$ can be unfaithfully mimicked for two reasons: either the dotted letter (representing the position of the head) is not transferred properly (first automaton below), or the rest of the configuration is not copied properly (second automaton below).



All other cases can be handled in a similar way, which concludes the proof. □

This lower bound applies as well for the inclusion problem in the very same classes of timed automata. Note that strongly non-Zeno timed automata are never universal, but we can modify the above proof to show that the inclusion problem is EXPSPACE-hard as well for strongly non-Zeno timed automata.

**Summary of the results.** We can summarize our results in the following table. The column on the left indicates the subclass we consider. New results are in black and italic, and in particular we can notice that there was no lower bound known for the class $\mathsf{IRTA}$.

| | size of the det. TA | universality problem | inclusion problem |
|---|---|---|---|
| $\mathsf{TA}_p$ | *doubly exp.* | *EXPSPACE-complete* | *EXPSPACE-complete* |
| $\mathsf{SnZTA}$ | *doubly exp.* | trivial | *EXPSPACE-complete* |
| $\mathsf{ECTA}$ [3] | exp. | PSPACE-complete | PSPACE-complete |
| $\mathsf{IRTA}$ [9] | doubly exp. | EXPSPACE-*complete* | EXPSPACE-*complete* |

## 5 Conclusion

In this paper, we proposed a general framework for the determinization of timed automata by means of an infinite timed tree. We showed that for a wide range of timed automata this infinite tree is language-equivalent to a deterministic timed automaton. The construction of this deterministic timed automaton yields the basis for algorithms to check universality or language inclusion. Concerning the complexity, these algorithms applied to event-clock timed automata [3] and timed automata with integer resets [9] provide tight bounds. In addition, our general framework yields the decidability of the universality problem for strongly non-Zeno timed automata, which was not known before.

We have focused on finite timed words, but we believe the procedure can be extended to timed automata over infinite timed words (with an $\omega$-regular acceptance condition), by incorporating a Safra-like construction in our procedure. In that framework the strong non-Zenoness assumption will even make more sense, and we thus claim that strongly non-Zeno timed automata are determinizable!

## References

1. Adams, S., Ouaknine, J., Worrell, J.: Undecidability of universality for timed automata with minimal resources. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 25–37. Springer, Heidelberg (2007)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Alur, R., Fix, L., Henzinger, T.A.: A determinizable class of timed automata. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 1–13. Springer, Heidelberg (1994)
4. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: Proc. IFAC Symposium on System Structure and Control, pp. 469–474. Elsevier Science, Amsterdam (1998)
5. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T.: When are timed automata determinizable? Research Report LSV-09-08, Laboratoire Spécification & Vérification, ENS de Cachan, France (2009)
6. Finkel, O.: Undecidable problems about timed automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 187–199. Springer, Heidelberg (2006)
7. Ouaknine, J., Worrell, J.: On the language inclusion problem for timed automata: Closing a decidability gap. In: Proc. 19th Annual Symposium on Logic in Computer Science (LICS 2004), pp. 54–63. IEEE Computer Society Press, Los Alamitos (2004)
8. Ouaknine, J., Worrell, J.: On the decidability and complexity of metric temporal logic over finite words. Logical Methods in Computer Science 3(1:8) (2007)
9. Suman, P.V., Pandya, P.K., Krishna, S.N., Manasa, L.: Timed automata with integer resets: Language inclusion and expressiveness. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 78–92. Springer, Heidelberg (2008)