Let's Be Lazy, We Have Time
or, Lazy Reachability Analysis for Timed Automata

Loïg Jezequel[1] and Didier Lime[2]

[1] Université de Nantes, LS2N UMR CNRS 6004
loig.jezequel@ls2n.fr
[2] École Centrale de Nantes, LS2N UMR CNRS 6004
didier.lime@ec-nantes.fr

Abstract. In a recent work we proposed an algorithm for reachability analysis in distributed systems modeled as networks of automata. The main interest of this algorithm is that it performs its analysis in a lazy way: decision is done by only taking into account the automata potentially involved in a path to the reachability goal. This new work extends the approach to networks of timed automata, lazily considering the automata but also lazily adding the clocks to the analysis, which implies not only to consider clocks tested along the paths to the goal, but also to deal with the special issues due to urgency and shared clocks. We have implemented our approach as a tool and provide some interesting experimental results, in a comparison with the model-checker Uppaal.

1 Introduction

The verification of concurrent timed systems is a crucial and challenging issue. It is subject to both an explosion of the number of discrete states due to the number of concurrent components, and an explosion of the clock space due to the number of clocks.

To model such systems we focus here on networks of timed automata and the verificaion of reachability properties. Since the seminal article of Alur and Dill establishing the PSPACE completeness of this problem [1], many techniques have been developed to improve the practical efficiency of reachability verification.

Efficient symbolic representations of the clock space have been proposed, implemented using a data structure called Difference Bound Matrix [5, 7], as well as efficient algorithms to handle them [16]. Different kind of abstractions have then been devised to further improve them [2, 12].

Better exploration orders [13] have also been proposed and quite a few authors have defined partial-order reductions for timed automata (see e.g. [10, 18, 6] and the references therein). Some decision-diagram-based representations of the state-space have also been proposed [15, 20]. Several tools are available that implement part of these techniques, this is in particular the case of Uppaal [17].

In this article, we exploit a technique orthogonal to most of those mentioned above and build on a recently proposed lazy reachability analysis algorithm in compound systems modeled as networks of labelled transition systems [14], that

we extend to the timed setting. The resulting algorithm can be implemented using the successful DBM data structure. It starts separately from each of the components explicitly mentioned in the reachability property and adds other components, as well as clocks, on demand, based on the analysis of the successive overapproximations obtained by ignoring some components or clocks. Since we start by verifying separately that each component in the property can indeed reach its goal, the algorithm also performs synchronisations to make sure that in reaching its goal, one component does not prevent another one to do so.

The timed setting brings a few difficulties of its own, namely: choosing the clocks to add, handling urgency, and accounting for shared clocks. It can be proven that our algorithm is sound, complete and that it terminates. We provide a rather naive prototype implementation that nonetheless produce some interesting experimental results.

This paper is organized as follows. In Section 2 we recall basic definitions about timed automata, give some definitions and notations about compound systems built from timed automata, and we define the reachability problem we consider. In Section 3 we briefly recall the lazy reachability algorithm of [14] and we describe the modifications needed to make it cope with timed systems. In Section 4 we discuss an implementation of this algorithm as a publicly available tool: LaRA-T and we present an experimental evaluation, comparing its performances to those of Uppaal on a few classic examples.

2 Definitions

In this section we first recall standard definitions for timed automata and set the notations we use in the paper. Then, we define the reachability problem on compound systems built from timed automata that we aim at solving.

2.1 Timed automata

Definition 1 (Clocks, clock constraints). *Let X be a set of real-valued variables called* clocks*. A clock constraint over X is a constraint of the form $x \sim k$ with $x \in X$, $k \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$. The set of all possible such clock constraints over X is denoted by $\mathcal{B}(X)$. The subset of $\mathcal{B}(X)$ where $\sim \in \{<, \leq\}$ is denoted by $\mathcal{B}'(X)$.*

For simplicity, we do not consider clock differences in the above defined constraints. The high level algorithm presented in this paper is however independant of the exact reachability analysis technique used, so our approach is not restricted to these simple constraints.

Definition 2 (Clock valuations). *For a set of clocks X we call* clock valuation *a function $v : X \to \mathbb{R}_{\geq 0}$ associating a non-negative real number to each clock in X. We denote by $V(X)$ the set of all such valuations. Given a subset R of X and a clock valuation v, we denote by $v[R]$ the clock valuation such that $\forall x \in R, v[R](x) = 0$ and $\forall x \notin R, v[R](x) = v(x)$. Given $d \in \mathbb{R}_{\geq 0}$, we denote by $(v + d)$ the clock valuation such that $\forall x \in X, (v + d)(x) = v(x) + d$.*

Definition 3 (Constraint satisfaction). *A clock valuation v satisfies a constraint $b = x \sim k$, written $v \models b$, if and only if $v(x) \sim k$. A clock valuation v satisfies a set of constraints g, written $v \models g$, if and only if it satisfies each of its elements.*

In this article, we consider timed automata with invariants [11]. Finite automata are extended with clocks that can be tested in guards to allow some transition to be taken, and can also be reset to 0 when some transition is taken. Invariants further specify constraints on clocks that must be satisfied to stay or enter in a given location. They add a notion of *urgency* to the timed automata of [1].

Definition 4 (Timed automata: syntax). *A* Timed automaton *(TA) is a tuple $\mathcal{A} = (\mathcal{L}, \ell^0, \Sigma, X, E, Inv)$ where $\mathcal{L}$ is a set of* locations*, $\ell^0 \in \mathcal{L}$ is an* initial location*, Σ is a set of* action labels*, X is a set of clocks, $E \subseteq \mathcal{L} \times 2^{\mathcal{B}(X)} \times \Sigma \times 2^X \times \mathcal{L}$ is a set of* transitions*, and $Inv : \mathcal{L} \to 2^{\mathcal{B}'(X)}$ associates* invariants *to the locations.*

In the rest of the paper $\mathcal{A}$ will denote the tuple $(\mathcal{L}, \ell^0, \Sigma, X, E, Inv)$. Similarly, $\mathcal{A}'$ will denote the tuple $(\mathcal{L}', \ell^{0\prime}, \Sigma', X', E', Inv')$. And, for any i, $\mathcal{A}_i$ will denote the tuple $(\mathcal{L}_i, \ell_i^0, \Sigma_i, X_i, E_i, Inv_i)$. For a transition $e = (\ell, g, \sigma, R, \ell') \in E$, we note $\ell(e) = \ell$, $g(e) = g$, $\sigma(e) = \sigma$, $R(e) = R$, and $\ell'(e) = \ell'$. Moreover, $\Sigma(\mathcal{A}) = \{\sigma \in \Sigma \ : \ \exists e \in E, \sigma(e) = \sigma\}$ denotes the set of actions actually used by $\mathcal{A}$. Similarly, $X(\mathcal{A}) = \{x \in X \ : \ \exists e \in E, g(e) \cap \mathcal{B}(\{x\}) \neq \emptyset \ \vee \ x \in R(e)\} \cup \{x \in X \ : \ \exists \ell \in \mathcal{L}, Inv(\ell) \cap \mathcal{B}(\{x\}) \neq \emptyset\}$ denotes the set of clocks actually appearing in $\mathcal{A}$. And finally, $\mathsf{Res}(\mathcal{A}) = \{x \in X \ : \ \exists (\ell, g, \sigma, R, \ell') \in E, x \in R\}$ denotes the set of clocks reset by $\mathcal{A}$.

A state of a timed automaton consists of a location, and a value for each of its clocks. The state of a timed automaton evolves either by letting time pass, or by taking a transition. This is formalised in the following definition:

Definition 5 (Timed automata: semantics, timed-paths, timed-runs, reachable locations, duration). *A* state *of a timed automaton $\mathcal{A}$ is a pair $(\ell, v) \in \mathcal{L} \times V(X)$ so that $v \models Inv(\ell)$. A* transition relation *$\to_{\mathcal{A}}$ is defined over the states of $\mathcal{A}$ as follows: $(\ell, v) \to_{\mathcal{A}} (\ell', v')$ if and only if:*

- *$\ell = \ell'$ and $\exists d \in \mathbb{R}_{\geq 0}$ so that $v' = (v + d)$ (time elapsing of duration d), or*
- *$\exists e \in E$, such that $\ell(e) = \ell$, $\ell'(e) = \ell'$, $v \models g(e)$, and $v' = v[R(e)]$ (discrete transition firing – of duration 0).*

A finite timed-path *(or simply, path) of $\mathcal{A}$ is a sequence $(\ell_0, v_0) \dots (\ell_n, v_n)$ of states such that $\forall i \in \{0..n-1\}, (\ell_i, v_i) \to_{\mathcal{A}} (\ell_{i+1}, v_{i+1})$. If, moreover, $\ell_0 = \ell^0$ (i.e ℓ_0 is the initial location of $\mathcal{A}$), we call $(\ell_0, v_0) \dots (\ell_n, v_n)$ a* finite timed-run *(or simply, run) of $\mathcal{A}$. When there exists such a run, we say that ℓ_n is* reachable *in $\mathcal{A}$. The* duration *of a timed-path is the sum of the durations of its transitions.*

In the rest of the paper we denote by $\mathsf{Urg}(\mathcal{A})$ the set of initially urgent locations of $\mathcal{A}$. These locations are the ones appearing on a timed-run so that

any location in it has a non-empty invariant. Formally, $\mathsf{Urg}(\mathcal{A})$ is the smallest set such that (a) if $Inv(\ell^0) \neq \emptyset$ then $\ell^0 \in \mathsf{Urg}(\mathcal{A})$, and (b) if $e \in E$ with $\ell(e) \in \mathsf{Urg}(\mathcal{A})$ and $Inv(\ell'(e)) \neq \emptyset$ then $\ell'(e) \in \mathsf{Urg}(\mathcal{A})$.

And we denote by $\mathsf{Req}(\mathcal{A})$ the set of actions that could be requested by $\mathcal{A}$ due to urgency: the actions appearing in transitions originating from locations in $\mathsf{Urg}(\mathcal{A})$. Formally, $\mathsf{Req}(\mathcal{A}) = \{\sigma \in \Sigma \ : \ \exists e \in E, \sigma(e) = \sigma \ \wedge \ \ell(e) \in \mathsf{Urg}(\mathcal{A})\}$.

2.2 Reachability problem in compound systems

We are interested in systems made of several interacting components. To formalise this notion, we define compound systems.

Definition 6 (Compound system). *Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be TAs. The* compound system *$\mathcal{A}_1 || \ldots || \mathcal{A}_n$ is the TA $\mathcal{A}$ such that:*

- $\mathcal{L} = \mathcal{L}_1 \times \ldots \times \mathcal{L}_n$, $\ell^0 = (\ell^0_1, \ldots, \ell^0_n)$, $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$, $X = X_1 \cup \ldots \cup X_n$,
- $((\ell_1, \ldots, \ell_n), g_1 \cup \ldots \cup g_n, \sigma, R_1 \cup \ldots \cup R_n, (\ell'_1, \ldots, \ell'_n)) \in E$ *if and only if* $\forall i \in [1..n]$*:*
  - $\sigma \in \Sigma_i$ *implies* $(\ell_i, g_i, \sigma, R_i, \ell'_i) \in E_i$*, and*
  - $\sigma \notin \Sigma_i$ *implies* $\ell_i = \ell'_i$*,* $g_i = \emptyset$*, and* $R_i = \emptyset$*,*
- $\forall \ell = (\ell_1, \ldots, \ell_n) \in \mathcal{L}$*,* $Inv(\ell) = Inv_1(\ell_1) \cup \ldots \cup Inv_n(\ell_n)$*.*

Definition 7 (Time-non blocking timed automaton). *A timed automaton $\mathcal{A}$ is* time-non blocking *if, in any state and for any duration d, there exists a finite timed-path from that state with duration d.*

In all this article, we assume that compound systems are time-non blocking TA. Else, an invariant of any TA could block the whole system. This would force to always consider all the TA of a system to perform reachability analysis.

The next definitions allow us to specify reachability objectives related to only a part of the global system.

Definition 8 (Global locations, partial locations, concretisation). *In a compound system $\mathcal{A}_1 || \ldots || \mathcal{A}_n$ we call any element from $\mathcal{L}_1 \times \ldots \times \mathcal{L}_n$ a* global location*. We call* partial location *any element from $(\mathcal{L}_1 \cup \{\star\}) \times \ldots \times (\mathcal{L}_n \cup \{\star\}) \setminus \{(\star, \ldots, \star)\}$, where $\star$ is a special symbol not in any $\mathcal{L}_i$. We say that a global location $(\ell'_1, \ldots, \ell'_n)$* concretises *the partial location $(\ell_1, \ldots, \ell_n)$ if and only if $\forall i \in [1..n]$, $\ell_i \neq \star$ implies $\ell'_i = \ell_i$.*

Definition 9 (Reachability problem). *In a compound system $\mathcal{A}$ we say that a partial location ℓ is reachable (in $\mathcal{A}$) if and only if there exists a global location that (1) is reachable in $\mathcal{A}$ and (2) concretises ℓ. Given a set $\mathcal{R}$ of partial locations, we call* reachability problem *the problem of deciding if there exists a reachable element in $\mathcal{R}$. We denote this problem by $RP^{\mathcal{R}}_{\mathcal{A}}$.*

In this paper, we propose a lazy backtracking-based algorithm for solving such reachability problems. We avoid as much as possible to compute the full compound systems considered, only taking into account the subsets of their components and clocks that are needed for reachability analysis. In the remaining of this section we introduce a few more definitions, simplifying the description of our algorithm.

2.3 Partial compound systems

The following definitions allow us to reason about only parts of the global system, be they obtained by considering only some of the components, some of the clocks, or even a partial behaviour of some components.

Definition 10 (Isomorphic timed automata). *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two TA. We say that they are* isomorphic *if and only if $\Sigma_1 = \Sigma_2$, $X_1 = X_2$, and there exists a bijection $f : \mathcal{L}_1 \to \mathcal{L}_2$ so that: $f(\ell_1^0) = \ell_2^0$, $(\ell, g, \sigma, R, \ell') \in E_1$ if and only if $(f(\ell), g, \sigma, R, f(\ell')) \in E_2$ and $\forall \ell \in \mathcal{L}_1, Inv_1(\ell) = Inv_2(f(\ell))$.*

Definition 11 (Neutral element). *We denote by $\mathcal{A}_{id}$ the TA such that $\mathcal{L}_{id} = \{id\}$, $\ell_{id}^0 = id$, $\Sigma_{id} = \emptyset$, $X_{id} = \emptyset$, $E_{id} = \emptyset$, and $Inv_{id}(id) = \emptyset$. As, for any TA $\mathcal{A}$, $\mathcal{A}||\mathcal{A}_{id}$ is isomorphic to $\mathcal{A}$, $\mathcal{A}_{id}$ can be considered as the neutral element for the composition of TAs. For any TA $\mathcal{A}$ we denote by $id(\mathcal{A})$ the TA whose only location is the initial location of $\mathcal{A}$ and which is isomorphic to $\mathcal{A}_{id}$.*

Definition 12 (Clock projection). *For a TA $\mathcal{A}$ and a set of clocks X', we denote by $P_{X'}(\mathcal{A})$ the TA $\mathcal{A}'$ so that: $\mathcal{L}' = \mathcal{L}$, $\ell^{0\prime} = \ell^0$, $\Sigma' = \Sigma$, X' is the set of clocks, $E' = \{(\ell, g \cap \mathcal{B}(X'), \sigma, R \cap X', \ell') \ : \ (\ell, g, \sigma, R, \ell') \in E\}$, Inv' is so that $\forall \ell \in \mathcal{L}, Inv'(\ell) = Inv(\ell) \cap \mathcal{B}(X')$.*

Definition 13 (Extensions). *A TA $\mathcal{A}_1$* extends *a TA $\mathcal{A}_2'$, noted $\mathcal{A}_1 \sqsupseteq \mathcal{A}_2'$, if and only if $\mathcal{A}_2'$ is isomorphic to some TA $\mathcal{A}_2$ so that: $\mathcal{L}_2 \subseteq \mathcal{L}_1$, $\ell_2^0 = \ell_1^0$, $\Sigma_2 \subseteq \Sigma_1$, $X_2 = X_1$, $E_2 \subseteq E_{1|\mathcal{L}_2,\Sigma_2}$, $Inv_2 = Inv_{1|\mathcal{L}_2}$, with $E_{1|\mathcal{L}_2,\Sigma_2} = \{e \in E_1 \ : \ \ell(e), \ell'(e) \in \mathcal{L}_2 \wedge \sigma(e) \in \Sigma_2\}$ and $Inv_{1|\mathcal{L}_2}$ the function defined over $\mathcal{L}_2$ and so that $\forall \ell \in \mathcal{L}_2, Inv_{1|\mathcal{L}_2}(\ell) = Inv_1(\ell)$. If moreover at least one of the above inclusions is strict, we say that $\mathcal{A}_1$* strictly extends *$\mathcal{A}_2'$, which we note $\mathcal{A}_1 \sqsupset \mathcal{A}_2'$.*

Definition 14 (Initialisation). *For a TA $\mathcal{A}$ and a set of clocks X' we denote by $ini(\mathcal{A}, X')$ the TA with the same locations, initial location, and transitions as $id(\mathcal{A})$ but with X' as set of clocks, the same set of actions as $\mathcal{A}$, and the same invariants as $\mathcal{A}$ on its initial location. Notice that $P_{X'}(\mathcal{A}) \sqsupseteq ini(\mathcal{A}, X')$.*

Definition 15 (Partial compound system). *A TA $\mathcal{A}'$ is a* partial compound system *of $\mathcal{A} = \mathcal{A}_1||\dots||\mathcal{A}_n$ if there exist m TAs $\mathcal{A}'_{k_1}, \dots, \mathcal{A}'_{k_m}$ with $\{k_1, \dots, k_m\} \subseteq [1..n]$, such that $\mathcal{A}' = \mathcal{A}'_{k_1}||\dots||\mathcal{A}'_{k_m}$ and $P_{X'_{k_i}}(\mathcal{A}_{k_i}) \sqsupseteq \mathcal{A}'_{k_i}$ for all i in $[1..m]$.*

In the rest of this paper, for a compound system $\mathcal{A} = \mathcal{A}_1||\dots||\mathcal{A}_n$, a partial compound system $\mathcal{A}' = \mathcal{A}'_{k_1}||\dots||\mathcal{A}'_{k_m}$ of $\mathcal{A}$, the set $K = \{k_1, \dots, k_m\}$, and a set $\mathcal{R}$ of partial locations of $\mathcal{A}$, we adopt the following notations.

By $\mathcal{A}' \to \mathcal{R}_{|K}$ we denote that $\mathcal{R}$ is (partially) reachable in $\mathcal{A}'$. That is, the fact that there exists a location from $\mathcal{R}_{|K} = \{(\ell'_{k_1}, \dots, \ell'_{k_m}) \ : \ \exists (\ell_1, \dots, \ell_n) \in \mathcal{R}, \forall i \in [1..n], \forall j \in [1..m], i = k_j \Leftrightarrow \ell_i = \ell'_{k_j}\}$ which is reachable in $\mathcal{A}'$.

By $\mathsf{Conf}(\mathcal{A}', K)$ we denote the set of actions in conflict with $\mathcal{A}'$ with respect to K: the actions from $\mathcal{A}_K = \mathcal{A}_{k_1}||\dots||\mathcal{A}_{k_m}$ originating from locations in $\mathcal{L}'$ but not appearing in transitions from E'. Formally $\mathsf{Conf}(\mathcal{A}', K) = \{\sigma \notin \Sigma(\mathcal{A}') \ : \ \exists e_K \in E_K, \sigma(e_K) = \sigma \ \wedge \ \ell(e_K) \in \mathcal{L}'\}$ (assuming not only isomorphism but equality in Definition 13).

3 From lazy reachability to lazy reachability with time

In this section we give an as generic as possible description of our lazy reachability algorithm. This description is strongly based on what we proposed recently for non-timed systems [14]. We show that it extends naturally to timed-systems. This however implies non-trivial modifications, in particular to handle invariants and resets of shared clocks. These modifications mostly impact the notion of *completeness*, which is the main notion behind the validity of the approach.

3.1 Introductory example

The main goal of our algorithm is to be as lazy as possible when solving a reachability problem in a compound system. That is, trying to involve the smallest number of automata and the smallest number of clocks of these automata in the reachability analysis. In order to get an overview of our approach to do so, lets consider the example of Figure 1.

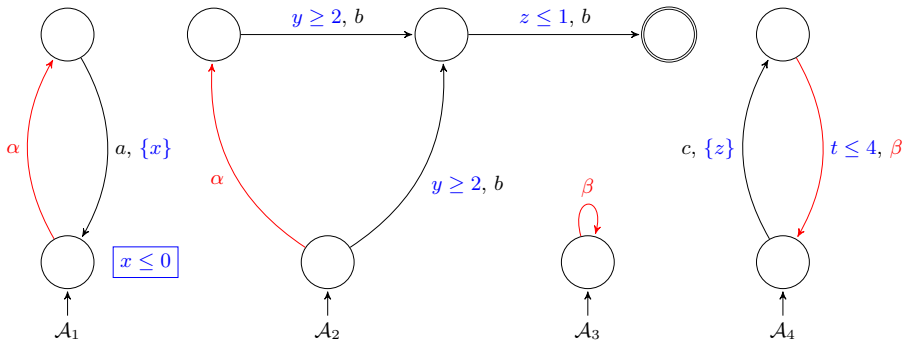

Fig. 1. A compound system involving four timed automata.

This figure shows a compound system made of four TAs: $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, and $\mathcal{A}_4$. Locations are depicted as circles and transitions as arrows with labels indicating guards, actions, and clocks resets (in this order, with empty guards and empty sets of resets omitted). Initial locations are the ones with input arrows coming from no other location. Invariants are depicted in rectangles near the corresponding locations (there is only one invariant, in $\mathcal{A}_1$, in this figure). There are three interactions in this system: $\mathcal{A}_1$ interacts with $\mathcal{A}_2$ because they share the α action, $\mathcal{A}_2$ interacts with $\mathcal{A}_4$ because they share the z clock, and $\mathcal{A}_3$ interacts with $\mathcal{A}_4$ because they share the β action. The objective here is to reach the double circled location in $\mathcal{A}_2$.

One can start from a partial compound system $P_\emptyset(\mathcal{A}_2)$, and look for a path to the objective. A possibility is bb. The clock constraints have to be added, moving to a partial compound system $\mathcal{A}_2$. The constraint $y \geq 2$ for the first b

can be satisfied by waiting for (at least) 2 time units. After that, the constraint $z \leq 1$ for the second b cannot be fulfilled. However, z is a clock that can be reset by $\mathcal{A}_4$. One thus adds $P_{\{z\}}(\mathcal{A}_4)$ to the partial compound system. It appears that bcb allows to reach the objective, fulfilling all the timing constraints.

However, in $\mathcal{A}_2$, the first b is in conflict with α. And α is shared with $\mathcal{A}_1$, where it has to be used with no delay, due to the invariant $x \leq 0$. This immediately discards bcb as a possible path to the reachability objective in the global compound system. One thus, looks for another path to the objective in $\mathcal{A}_2||P_{\{z\}}(\mathcal{A}_4)$.

A possibility is αbcb. This immediately implies to add $\mathcal{A}_1$ to the partial compound system because of the shared action α. In the partial compound system $\mathcal{A}_1||\mathcal{A}_2||P_{\{z\}}(\mathcal{A}_4)$, αbb is clearly a timed-run. It can be verified that this run is also a run of the global compound system.

Using this incremental process, one has found a way to satisfy the reachability objective. This has been done without considering the automaton $\mathcal{A}_3$, nor the clock t. This is why we call our algorithm lazy. The remainder of this section formalizes the approach we just exemplified.

3.2 General scheme of the algorithm

Algorithm 1 presents the general scheme of our algorithm[3]. This algorithm starts from a partition of the TAs involved in the reachability objective. The idea is to verify this objective separately on each involved component, with the hope to find a solution involving no interaction between these components. The current state of the search is stored in the list Ls which has initially one element per part of the initial partition. Each such element is a list of tuples (A, C, I, J, K, L, M) – described in details in the next subsection – that represent more and more concrete partial compound systems: they include more and more automata and take into account more and more clocks. The more concrete partial compound system is at the head of the list.

First note that the initial partition of the TAs has to be well-formed (line 1), by that we mean that it should contain only one element $I_1 = \mathbb{L}_g$ if any TA has an invariant on a reachability objective (that is $\exists i \in \mathbb{L}_g$, $\exists(\ell_1, \ldots, \ell_n) \in \mathcal{R}$, so that $\ell_i \neq \star$ and $Inv_i(\ell_i) \neq \emptyset$).

Each element in the initial partition has to be completed with all automata resetting some clocks of the automata in that element (lines 3,4). Resetting a clock may indeed add some behaviours, which could be overlooked otherwise.

After initialising Ls, we proceed to the main loop, which consists of two operations: concretisation, ensuring completeness, and merging, ensuring consistency. These notions and functions are described in the following subsections.

[3] Notice that the algorithms presented in this paper make use of the classic abstract list data-structure. The usual operations $hd()$, $tl()$, and $len()$ give respectively the *head*, *tail*, and *length* (number of elements) of a list. The list constructor (prepend) is denoted by : and the list concatenation is denoted by ++ . The $rev()$ operator reverses a list. The empty list is denoted by [ ]. Finally, $L[i]$ denotes the i^{th} element of list L.

Algorithm 1 Algorithm solving $RP^{\mathcal{R}}_{\mathcal{A}}$ ($\mathbb{L}_g$: indices of TAs involved in $\mathcal{R}$)

```
1  choose any well-formed partition {I_1, ..., I_p} of L_g
2  for all k in [1..p] {
3    let I'_k = I_k
4    until stability  let I'_k = I'_k ∪ {i ∉ I'_k : ∃j ∈ I'_k, Res(A_i) ∩ X(A_j) ≠ ∅}
5    let ID_k = ||_{i∈I'_k} id(A_i)
6    let INI_k = ||_{i∈I'_k} ini(A_i, ∅)
7  }
8  let Ls = [[(ID_1, INI_1, I_1, ∅, I'_1, ∅, ∅)], ..., [(ID_p, INI_p, I_p, ∅, I'_p, ∅, ∅)]]
9  let Complete = false
10 let Consistent = false
11 while not Complete or not Consistent {
12   let Complete = IsComplete(Ls)
13   if not Complete {
14     optional unless Consistent {
15       if not Concretise(Ls) { return false }
16     }
17   }
18   let Consistent = IsConsistent(Ls)
19   if not Consistent {
20     optional unless Complete { Merge(Ls) }
21   }
22 }
23 return true
```

3.3 Completeness, concretisation and the Concretise function

The main loop of Algorithm 1 iterates as long as the current state Ls of the search is not *complete*. We say that Ls is complete (noted IsComplete(Ls)) if and only if, for any indice k, $Ls[k]$ is complete. Intuitively, this means that the partial compound system represented by $hd(Ls[k])$ contains a path to reach the corresponding local goal. Moreover, this path has to (1) use no action of any automaton not participating in this partial compound system, (2) have no conflict with actions that could be externally forced by an invariant (this was the case of action α in the above example), (3) avoid relying on the satisfaction of a clock constraint involving a clock that is reset by an automaton not from this partial compound system (this was the case of clock z in the above example), and (4) take into account all clock constraints on any transition involved. In other words: a partial compound system is complete if it can reach a local goal alone.

In any tuple (A, C, I, J, K, L, M), in particular in $hd(Ls[k])$, C is the TA representation of the partial compound system considered, and $J \cup K$ contains the indices of the TAs (from the global compound system) involved in this partial compound system. The above notion of completeness can be formalized from only C, J, and K. The other elements of the tuple are instrumental for building our

algorithm and are described later. In the following definition of completeness, points (1-4) correspond to intuitions (1-4) above.

Definition 16 (Completeness). *The list* $Ls[k]$ *so that* $hd(Ls[k]) = (A, C, I, J, K, L, M)$ *is complete if* $\exists C^*$ *so that:* $C \sqsupseteq C^*$, $C^* \rightarrow \mathcal{R}_{|J\cup K}$, *and (1)* $\forall i \notin J\cup K, \Sigma_i \cap \Sigma(C^*) = \emptyset$, *(2)* $\forall i \notin J\cup K, \mathsf{Conf}(C^*, J\cup K) \cap \mathsf{Req}(\mathcal{A}_i) = \emptyset$ *(3)* $\forall i \notin J\cup K, \mathsf{Res}(\mathcal{A}_i) \cap X(C^*) = \emptyset$ *(4)* $\{x \notin X(C^*) \ : \ \exists A^*, ||_{i\in J\cup K}\mathcal{A}_i \sqsupseteq A^*, P_{X(C^*)}(A^*) = C^*, x \in X(A^*)\} = \emptyset$.

In order to achieve completeness, we use the CONCRETISE function defined in Algorithm 2. This is basically a standard backtracking algorithm, specialized for incrementally building more and more concrete partial compound systems: partial compound systems containing (1) more and more states and transitions, for a fixed set of TAs and clocks, and (2) more and more TAs and clocks.

Algorithm 2 Auxiliary function CONCRETISE(Ls) for Algorithm 1

```
1  choose any k in [0..len(Ls) − 1] such that not IsComplete(Ls[k])
2  let (A, C, I, J, K, L, M) = hd(Ls[k])
3  let Back = tl(Ls[k])
4  if ∃C* s.t. (||_{i∈K} P_{L∪M}(A_i))||A ⊒ C* ⊐ C and C* → R_{|J∪K} {
5    choose any such C*
6    let N_A = {i ∉ J ∪ K : Σ_i ∩ Σ(C*) ≠ ∅}
7    let A' be such that ||_{i∈J∪K} A_i ⊒ A' and P_∅(A') = P_∅(C*)
8    let N_X = {x ∉ L ∪ M : x ∈ X(A')}
9    case N_A ∪ N_X ≠ ∅
10     choose any K', M' such that K' ⊆ N_A and M' ⊆ N_X and K' ∪ M' ≠ ∅
11     let Back' = (A, C*, I, J, K, L, M) : Back
12     let J' = J ∪ K
13     let L' = L ∪ M
14     let K'' = K'
15     until stability let K'' = {i ∉ J' ∪ K'' : ∃j ∈ K'', Res(A_i) ∩ X(A_j) ≠ ∅}
16     let A* be such that ||_{i∈J∪K} P_{L'∪M'}(A_i) ⊒ A* and P_∅(A*) = P_∅(C*)
17     let Ls[k] = (A*, ||_{i∈J'∪K''} ini(A_i, L' ∪ M'), I, J', K'', L', M') : Back'
18   case N = ∅
19     let Ls[k] = (A, C*, I, J, K, L, M) : Back
20 } else {
21   if Back = [ ] { return false }
22   else { let Ls[k] = Back }
23 }
24 return true
```

The list $Ls[k]$ contains the search history: backtracking means replacing this list by its tail (else part of the conditional at line 4). Notice that, when backtracking, it could be possible to allow to remember the unsuccessful searches (this can be used for speeding up the future searches). This has been described in [14] for un-timed systems.

Any element of $Ls[k]$, and in particular the one reflecting the current state of the search (the head of $Ls[k]$), is a tuple (A, C, I, J, K, L, M) where: A is the partial compound system computed at the previous call to CONCRETISE, C is the current partial compound system we consider, I gives the initial partition of TAs involved in the objective, J gives the TAs involved in A, K gives the TAs that can be used to build C from A, L gives the clocks involved in A, and M gives the clocks that are used to build C from A.

The main idea of the function consists in first choosing a partial compound system C^* bigger than what we already had, and that can reach the goal. Set $\mathcal{N}_\mathcal{A}$ then corresponds to the automata sharing some action with C^*, but not added, and $\mathcal{N}_X$ to the clocks tested, but not present, in C^*. From these sets, we can choose automata or clocks to add to our partial compound system in order to try to make it complete (line 10). The next lines create the new tuple that will be put at the head of Ls as the new current level of concretisation (line 17), to reflect these choices. Line 15 forces the addition of automata that reset some clocks we have chosen to add. This is important because resetting a clock may add some new behaviours. Finally, A^* (line 16) is a version of C^* with the whole set of clocks we have up to now (those we already had and those we have just chosen to add). It will serve as an upper bound (wrt. $\sqsupseteq$) for the choice of C^* at the next level of concretisation. Note that at this next level we start with a partial compound system reduced to the initial states of the chosen automata (and with all the chosen clocks). If we have no clocks or automata to add from $\mathcal{N}_X, \mathcal{N}_\mathcal{A}$, then we simply update the current partial compound with C^* (line 19) and back in the main algorithm we will check if this has allowed us to achieve completeness (line 11 of Algorithm 1). If we could not find C^* at all, then it is not possible to reach the goal with the current upper bound an we need to backtrack to extend this upper bound at a lower level of concretisation, i.e., with fewer clocks or automata (line 22).

3.4 Consistency, merging and the MERGE function

Achieving completeness for each element of Ls is not sufficient however to solve our reachability problem. Indeed, it may be the case that some automata appear in several different elements of Ls: we start from a partition but the same automata may be added to elements of Ls during concretisation steps. In this case, it is likely that some paths in different partial compound systems interfere: they use the same actions but not in the same order or they need incompatible valuations of the clocks for satisfying clock constraints. The main loop of Algorithm 1 reflects this by iterating as long as the current state Ls of the search is not complete, as explained above, or not *consistent*.

Definition 17 (Consistency). $Ls = [(A_1, C_1, I_1, J_1, K_1, L_1, M_1) : Back_1, \ldots, (A_n, C_n, I_n, J_n, K_n, L_n, M_n) : Back_n]$ *is* consistent *if* $\forall i \neq j \in [1..n], (J_i \cup K_i) \cap (J_j \cup K_j) = \emptyset$.

In order to achieve consistency we use a MERGE function to replace two elements of Ls: $Ls[i]$ (with $hd(Ls[i]) = (A_i, C_i, I_i, J_i, K_i, L_i, M_i)$ and $tl(Ls[i]) =$

$Back_i$) and $Ls[j]$ (with $hd(Ls[j]) = (A_j, C_j, I_j, J_j, K_j, L_j, M_j)$ and $tl(Ls[j]) = Back_j$) by a single one $h : Back$ obtained by merging them, thus reducing the length of Ls by one. The simplest such new element would be such that: $Back = [\,]$ and $h = (ID_i||ID_j, INI_i||INI_j, I_i \cup I_j, \emptyset, I'_i \cup I'_j, \emptyset, \emptyset)$. However, it may be of interest to use the current state of the search in both $Ls[i]$ and $Ls[j]$, taking instead: $Back = [(ID_i||ID_j, INI_i||INI_j, I_i \cup I_j, \emptyset, I'_i \cup I'_j, \emptyset, \emptyset)]$, and $h = (P_{L_j}(A_i)||P_{L_i}(A_j), P_{L_j \cup M_j}(C_i)||P_{L_i \cup M_i}(C_j), I_i \cup I_j, J_i \cup J_j, K_i \cup K_j)$.

In fact, it is even possible to replace $Ls[i]$ and $Ls[j]$ by any history that could have been produced by a sequence of call to Concretise starting from $(ID_i||ID_j, INI_i||INI_j, I_i \cup I_j, \emptyset, I'_i \cup I'_j, \emptyset, \emptyset)$ (of which the two above examples are particular cases). This avoids to un-merge when backtracking, has been formalized as a notion of good Merge in our previous work [14], and remains essentially the same for timed systems.

3.5 Termination, soundness, completeness

Theorem 1 explicits the fact that Algorithm 1 always terminates, and is sound and complete. Due to space limitations, its proof is omitted.

Theorem 1. *Algorithm 1 always terminates. It returns true if and only if the goal is reachable.*

3.6 Example

We get back to the introductory example and see how our algorithm may find the result we intuitively outlined.

First we choose an initial partition of the automata directly involved in the objective. Here there is only $\mathcal{A}_2$ so $Ls = [(id(\mathcal{A}_2)||id(\mathcal{A}_4), ini(\mathcal{A}_2, \emptyset)||ini(\mathcal{A}_4, \emptyset), \{2\}, \emptyset, \{2, 4\}, \emptyset, \emptyset)]$. Note in particular that $I'_1 = \{2, 4\}$ because clock z is present in $\mathcal{A}_2$ and reset by $\mathcal{A}_4$. Also, since we have only one element in our partition, we will always stay consistent in this example.

Now we can call Concretise and choose $k = 0$ (the only possibility). We choose for instance C^* as the compound system C_{24} made of the path bb in $\mathcal{A}_2$ and the path c in $\mathcal{A}_4$, which permits to reach the goal. Then $\mathcal{N}_\mathcal{A} = \emptyset$ and $\mathcal{N}_X = \{y, z\}$. We can now choose $K' = \emptyset$ and $M' = \{y, z\}$ for instance. Then $K'' = \emptyset$. Finally, A^* is $C^* = C_{24}$ augmented by the clocks y and z, slightly abusing notations we denote it by $P_{\{y,z\}}(C_{24})$. The function returns true, and $Ls[0] = [Ls_1, Ls'_0]$ with $Ls'_0 = (id(\mathcal{A}_2)||id(\mathcal{A}_2), C_{24}, \{2\}, \emptyset, \{2, 4\}, \emptyset, \emptyset)]$, and $Ls_1 = (P_{\{y,z\}}(C_{24}), ini(\mathcal{A}_2, \{y, z\})||ini(\mathcal{A}_4, \{y, z\}), \{2\}, \{2, 4\}, \emptyset, \emptyset, \{y, z\})$.

Back in the main algorithm, starting from $ini(\mathcal{A}_2, \{y, z\})||ini(\mathcal{A}_4, \{y, z\})$ we obviously do not have completeness. So we call Concretise again. We choose, for instance C^* as the full $P_{\{y,z\}}(C_{24})$, in which the goal can be reached. Both $\mathcal{N}_\mathcal{A}$ and $\mathcal{N}_X$ are empty, so we proceed to line 19 and $Ls[0]$ becomes $[(P_{\{y,z\}}(C_{24}), P_{\{y,z\}}(C_{24}), \{2\}, \{2, 4\}, \emptyset, \emptyset, \{y, z\}), Ls'_0]$. We return true.

Back in the main algorithm, $P_{\{y,z\}}(C_{24})$ is not complete because α is in conflict with the first b in $\mathcal{A}_2$ and α is urgent due to the invariant in $\mathcal{A}_1$ ((2) of

Definition 16) so we need to call CONCRETISE again. There, we have to choose something strictly bigger than the C, i.e $P_{\{y,z\}}(C_{24})$, which is not possible with K being empty. We reach line 22 (backtracking) and $Ls[0]$ becomes $[Ls'_0]$.

In the main algorithm again, Ls is not complete because C_{24} (the C value of Ls'_0) is not complete (for the same conflict with α as above). We choose to add the rest of $\mathcal{A}_2$ and choose therefore C^* as the compound system C'_{24} made of $\mathcal{A}_2$ and the path c of $\mathcal{A}_4$. Then $\mathcal{N}_\mathcal{A} = \{1\}$ because α is shared with $\mathcal{A}_1$ and $\mathcal{N}_X = \{y, z\}$. We choose $K' = \{1\}$, $M' = \emptyset$ and since we have no further shared clock, we have $K'' = K'$. Since we have not added clocks, $A^* = C'_{24}$ and finally $Ls[0]$ becomes $[Ls'_1, Ls''_0]$ with $Ls''_0 = (id(\mathcal{A}_2)||id(\mathcal{A}_4), C'_{24}, \{2\}, \emptyset, \{2, 4\}, \emptyset, \emptyset)$, and $Ls'_1 = (C'_{24}, ini(\mathcal{A}_1, \emptyset)||ini(\mathcal{A}_2, \emptyset)||ini(\mathcal{A}_4, \emptyset), \{2\}, \{2, 4\}, \{1\}, \emptyset, \emptyset)$. We return true.

Back in the main algorithm we are not complete since we do not reach the goal in $ini(\mathcal{A}_1, \emptyset)||ini(\mathcal{A}_2, \emptyset)||ini(\mathcal{A}_4, \emptyset)$ so we call CONCRETISE. We choose for instance C^* as the compound system C_{124} made of the path α in $\mathcal{A}_1$, the whole of $\mathcal{A}_2$ and the path c in $\mathcal{A}_4$. Then $\mathcal{N}_\mathcal{A} = \emptyset$ but $\mathcal{N}_X = \{x, y, z\}$ because clock x is tested in the invariant of $\mathcal{A}_1$. So we have to choose $K' = \emptyset$ and decide to take $M' = \{x, y, z\}$. We have $K'' = K'$. Hence, A^* is C_{124} augmented by the clocks x, y and z, slightly abusing notations we denote it by $P_{\{x,y,z\}}(C_{124})$. And $Ls[0]$ becomes $[Ls_2, Ls''_1, Ls''_0]$, with $Ls''_1 = (C'_{24}, C_{124}, \{2\}, \{2, 4\}, \{1\}, \emptyset, \emptyset)$ and $Ls_2 = (P_{\{x,y,z\}}(C_{124}),\ ini(\mathcal{A}_1, \{x, y, z\})\ ||\ ini(\mathcal{A}_2, \{x, y, z\})\ ||\ ini(\mathcal{A}_4, \{x, y, z\}),\ \{2\},\ \{1, 2, 4\}, \emptyset, \emptyset, \{x, y, z\})$. We return true.

Again in the main algorithm we are not complete because we do not reach the goal in the C value of Ls_2, so we call CONCRETISE. We choose C^* as the compound system C'_{124} made of the path α in $\mathcal{A}_1$ (but with x this time), the whole of $\mathcal{A}_2$ (with y and z this time) and the path c in $\mathcal{A}_4$ (with z this time). Now taking into account the clocks C'_{124} does allow to reach the goal, but only through the path αbcb. Furthermore, $\mathcal{N}_X = \mathcal{N}_\mathcal{A} = \emptyset$ so we proceed to line 19 and $Ls[0]$ is now $[Ls'_2, Ls''_1, Ls''_0]$, with $Ls'_2 = (P_{\{x,y,z\}}(C_{124}), C'_{124}, \{2\}, \{1, 2, 4\}, \emptyset, \emptyset, \{x, y, z\})$. We return true. Finally, the main algorithm concludes that C'_{124} is complete and terminate, by concluding that the goal is indeed reachable.

4 Experimental analysis

We implemented an instance of the above algorithm as a tool called LARA-T. The LARA-T tool represents around 2000 lines of code written in the functional language Haskell[4]. It is built over the previous LARA tool for reachability analysis in networks of (untimed) automata [14]. Both tools are available for download[5].

The algorithm we have presented is very general and the current implementation results from some important choices. First, each time we add automata or clocks, we compute the resulting partial compound system completely. This

[4] https://www.haskell.org/
[5] http://lara.rts-software.org/

eliminates the need for backtracking, since if we cannot find the goal in this partial compound system then it is for sure not reachable at all. In that respect, this choice also heuristically favors the unreachable case. Second, we only compute the reachable parts of the compound systems, using a classic DBM-based symbolic state exploration [16], with DBM inclusion checking and maximal constant per clock extrapolation. Finally, when we have computed a whole partial compound system, we look at the paths we have followed to reach all goal states. If some automata or clocks are required for completeness with respect to all those paths, then we add them all at the next level. Otherwise, we add one arbitrary required automaton or clock with the following priority: first automata ((1) of Definition 16), then clocks actually used on some path to the goal (4), and finally clocks coming from urgent conflicts due to invariants (2). We have not yet implemented the support for shared clocks (hence (3) of Def. 16 not appearing in the previous priority order).

We compared LaRA-T to Uppaal [4] on several examples from the literature on analysis and verification of timed systems[6]:

- CritReg is a modeling of a critical region protocol from PAT [19] benchmarks. We check the reachability of an error location in one of the automata. This is a true property.
- Fddi is a modeling of a communication protocol based on a token ring network presented in [9] and adapted to our setting in [13]. We check the reachability of a couple of mutually exclusive locations. This is a false property.
- Fischer is a modeling of Fischer's mutual exclusion protocol [3]. We check the reachability of a couple of mutually exclusive locations. This is a false property.
- Fischer2 is a variation of Fischer's mutual exclusion protocol where a time constant has been changed, breaking the mutual exclusion guarantee. In this example, the property is true.
- Trains1 is the Train model of Uppaal [3]. A controller is responsible for a queue of trains and has to prevent more than one of them to access a bridge together. We check the reachability of a state in which the two first trains are crossing the bridge at the same time. This is a false property.
- Trains2 is a variation of the Trains1 benchmark where the controller manages a set of trains rather than a queue of trains.
- Trains3 is a model where a railway crosses a road [8]. A controller has to ensure that the gate is closed (cars cannot crosse the railway) as soon as a train crosses the road. We check the reachability of a state in which a train crosses the road while the gate is open. This is a false property.

4.1 Experimental results

For each example we ran Uppaal and LaRA-T on instances of growing size (the size is, roughly, the number of timed-automata, see Table 2 for precise

[6] Uppaal templates and inputs for LaRA-T: http://lara.rts-software.org/

information), with a time-limit of 20 minutes at each size. In order to better evaluate the performances of the laziness mechanism (independently of our implementation of the DMB-based computation of the state-space of a TA) we also ran LaRA-T with all components and clocks in a single element of the initial partition (LaRA-T Full). All the experiments were run on the same machine with four Intel® Xeon® E5-2620 processors (six cores each) with 128GB of memory. Though this machine has some potential for parallel computing, all the experiments presented here are actually monothreaded.

Table 1 gives the largest instance of each example solved by each tool within the time-limit.

Table 1. Size of last instances solved within 20 minutes by Uppaal and LaRA-T

	CritReg	Fddi	Fischer	Fischer2	Trains1	Trains2	Trains3
LaRA-T	≥ 1500	≥ 5000	7	≥ 500	8	13	7
LaRA-T Full	4	15	6	5	8	13	5
Uppaal	46	13	13	65	10	16	6

For each example, we also evaluated the number of automata and clocks that LaRA-T takes into account to decide its result in an instance of size n. In Table 2, we compare it to the total number of automata and clocks in the same instance. Our goal was to evaluate to which extent our algorithm is actually lazy.

Table 2. Number of automata (A) and clocks (C) used by LaRA-T for solving instances of size n, and total number of automata and clocks in such instances.

	CritReg		Fddi		Fischer		Fischer2		Trains1		Trains2		Trains3	
	A	C	A	C	A	C	A	C	A	C	A	C	A	C
LaRA-T	3	1	4	5	$n+1$	n	3	2	3	3	3	3	$n+2$	3
total	$2n+1$	n	$n+1$	$3n+1$	$n+1$	n	$n+1$	n	$n+1$	n	$n+1$	n	$n+2$	$n+2$

4.2 Analysis of the results

We first remark that LaRA-T clearly outperforms Uppaal on three of our seven examples: CritReg, Fddi, and Fischer2. This is because the number of automata and clocks considered by LaRA-T in these examples does not increase with the size of the instances considered. This is particularly striking in the case of Fddi where the property we consider is false: Uppaal needs to completely explore a quickly growing state-space.

On three other examples, namely Trains1, Trains2, and Trains3, LaRA-T copes with Uppaal. It is surprising that, while the number of automata and clocks does not increase with the size of the instances considered, LaRA-T solves a bit less instances of Trains1 and Trains2 than Uppaal. This can be

explained by the fact that one of the automata considered by LaRA-T represents the centralized data-structure (either a queue or a set) involved in these examples. The size of this automaton increases exponentially with the size of the instances considered. LaRA-T does not implement efficient search in large automata (this is orthogonal to our work). On the Trains3 example, LaRA-T always uses all the automata. However, it needs only a subset (of size independent from the size of the instance considered) of the clocks to conclude. So, the timed reachability analysis is made easier, explaining the fact that LaRA-T solves a bit more instances than Uppaal.

Finally, LaRA-T is clearly outperformed by Uppaal on the Fischer example. This can be explained by the fact that LaRA-T needs to consider all the automata and all the clocks, that is, to perform the full reachability analysis. On such a task it is illusory to be as efficient as Uppaal.

5 Conclusion

We have proposed a new algorithm for the verification of concurrent timed systems, modelled as products of timed automata. This algorithm extends our previous proposition for untimed systems [14] by considering not only the different automata components, but also clocks, in a lazy manner. By examining closely which actions and clocks are needed to reach some goal locations, and how they interact with urgency, in successive over-approximations of the whole system, we are, in many cases, able to conclude on the reachability property by considering only a subset of the components and clocks. And we can do it regardless of the actual truth value of the property. We have implemented a version of the algorithm in a freely available tool, named LaRA-T and we report on its efficiency in comparison to Uppaal, a state-of-the-art model-checker for timed automata. These experimental results, obtained on classic benchmarks from the timed systems community, are very encouraging, with LaRA-T outperforming Uppaal, sometimes by several magnitude orders, on a few of the benchmarks, and being never too far behind even in the worst cases where all components and clocks have to be considered to decide the property.

The proposed algorithm provides a quite general framework open for many heuristic improvements, and part of future work naturally consists in finding good heuristics for better choosing the components and clocks to add. The computed over-approximations also contain a lot of information on the system and, in practice, we currently only use them to prune actions leading to non-coreachable states. It would be interesting to make a better use of that information, and, for instance in the case of timed systems, to use it to cheaply find good exploration orders minimizing the number of "mistakes" when a bigger DBM is reached after a smaller one for a given location, and all the successors have to be explored again (see [13] for precise account of the problem). Further work also includes extending to properties beyond reachability, and taking discrete variables into account to improve the conciseness of the models.

References

1. Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *International Journal on Software Tools for Technology Transfer*, 8(3):204–215, June 2006.
3. Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In *Proceedings of Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 200–236, 2004.
4. Gerd Behrmann, Alexandre David, Kim G. Larsen, Paul Pettersson, and Wang Yi. Developing UPPAAL over 15 years. *Software - Practice and Experience*, 41(2):133–142, 2011.
5. Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.
6. Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *9th International Conference on Concurrency Theory*, pages 485–500, 1998.
7. Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time petri nets. In *IFIP Congress*, pages 41–46, 1983.
8. Bernard Berthomieu and François Vernadat. State class constructions for branching analysis of time petri nets. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 442–457, 2003.
9. Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool KRONOS. In *Proceedings of the DIMACS/SYCON Workshop Hybrid Systems III: Verification and Control*, pages 208–219, 1995.
10. Henri Hansen, Shang-Wei Lin, Yang Liu, Truong Khanh Nguyen, and Jun Sun. Diamonds are a girl's best friend: Partial order reduction for timed automata with abstractions. In *26th International Conference on Computer Aided Verification*, pages 391–406, 2014.
11. Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.
12. Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Information and Computation*, 251:67–90, 2016.
13. Frédéric Herbreteau and Thanh-Tung Tran. Improving search order for reachability testing in timed automata. In *13th International Conference on Formal Modeling and Analysis of Timed Systems*, volume 9884 of *Lecture Notes in Computer Science*, pages 124–139, Madrid, Spain, September 2015. Springer.
14. Loïg Jezequel and Didier Lime. Lazy reachability analysis in distributed systems. In *Proceedings of the 27th International Conference on Concurrency Theory*, pages 17:1–17:14, 2016.
15. Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.
16. Kim G. Larsen, Paul Pettersson, and Wang Yi. Model-checking for real-time systems. In *Fundamentals of Computation Theory*, pages 62–88, 1995.
17. Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Journal of Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

18. Ramzi Ben Salah, Marius Bozga, and Oded Maler. On interleaving in timed automata. In *17th International Conference on Concurrency Theory*, pages 465–476, 2006.
19. Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. Pat: Towards flexible verification under fairness. In *21th International Conference on Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer, 2009.
20. Farn Wang. Symbolic verification of complex real-time systems with clock-restriction diagram. In *Formal Techniques for Networked and Distributed Systems*, pages 235–250, 2001.