Automata Column

MAHESH VISWANATHAN, University of Illinois Urbana-Champaign
vmahesh@illinois.edu

For the Automata Theory Column, we have an article by B. Srivathsan on the reachability problem in timed automata. Ever since timed automata were introduced by Alur and Dill, they have been widely used in the verification of real-time systems. Their appeal comes from a simple definition that couples expressiveness with computational tractability. The control state reachability problem, which asks if there is a computation of a given timed automaton that reaches a given control state, is an important problem in this context and was shown to be decidable in PSPACE by Alur and Dill in their seminal paper. However, because of the central role this problem plays in verification, it has been extensively studied in the past 30 years. In this issue, Srivathsan lucidly articulates the main principles underlying modern approaches to solving this problem.

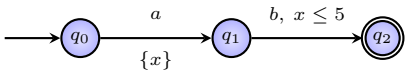
Reachability in timed automata

B Srivathsan, Chennai Mathematical Institute, India
CNRS IRL 2000, ReLaX, Chennai, India

Given a timed automaton $\mathcal{A}$ and a control state q, does there exist a run of $\mathcal{A}$ that visits q? This problem of control state reachability in timed automata was posed in [Alur and Dill 1994] and is known to be PSPACE-complete. One does not hope to have efficient algorithms for this problem, in theory. Nevertheless, research in this subject over the last three decades has led to industry-strength award-winning tools implementing this problem. This topic continues to be an active area of research even now. In this article, we present one successful algorithmic framework for attacking this problem.

1. INTRODUCTION

Timed automata [Alur and Dill 1994] are finite automata equipped with clocks. They recognize words with timestamps attached to each letter. For example, the automaton below recognizes $(a,t_1)(b,t_2)$ such that $t_2 - t_1 \leq 5$. Here, t_1 and t_2 are non-negative reals denoting the timestamp at which a and b occurred respectively. The symbol x denotes



a clock, which is assumed to start with value 0 in the initial state q_0. The notation $\{x\}$ means that x is reset to 0 when a is read. The constraint $x \leq 5$ denotes that letter b can be read only when the value of x is at most 5. This constraint is a guard on the transition. The reset of x at a and the guard $x \leq 5$ at b together imply $t_2 - t_1 \leq 5$, that is, the time between a and b is at most 5. The automaton could in general contain several clocks, have multiple constraints in a guard, and reset multiple clocks in a transition to express more complex timing constraints.

Timed automata can be used to model state-based systems that have timing constraints. Some illustrative examples include asynchronous circuits with gate delays [Maler and Pnueli 1995; Alur 1991], communication protocols [Daws and Tripakis 1998], mutual exclusion protocols [Alur et al. 1995] and scheduling problems [Abdeddaïm et al. 2006]. A comprehensive list of more involved case-studies can be found in the webpage of the tool UPPAAL[1], which has been the leading tool in timed automata verification. Detecting unwanted behaviours in such systems reduces to checking reachability in their timed automaton models. Therefore there is substantial interest in getting practically viable solutions for the reachability problem. This has led to several tools for the reachability analysis of timed automata: UPPAAL [Larsen et al. 1997], KRONOS [Daws et al. 1995], PAT [Sun et al. 2009], RED [Wang 2006], TChecker [Herbreteau and Point 2019], Theta [Tóth et al. 2017],

[1] https://uppaal.org/casestudies/

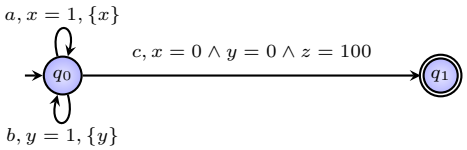


Fig. 1. Automaton $\mathcal{A}_1$ to illustrate a long witness for reachability.

LTS-Min [Kant et al. 2015], Symrob [Roussanaly et al. 2019], MCTA [Kupferschmid et al. 2008], etc.

We start with an example to illustrate the mechanics of timed automata and to also show why the reachability problem is difficult. Consider automaton $\mathcal{A}_1$ of Figure 1. It has two states and three clocks x, y, z. The goal is to reach q_1. Initially the automaton is at q_0, with all clocks being 0. Hence action c is not feasible initially. An elapse of δ time units at q_0 gives $x = y = z = \delta$, still not enabling c. To do c, the difference between x and z should be 100, the difference between y and z should be 100, and finally both x and y should be equal to 0. Doing a once gives $x = 0, y = 1, z = 1$: this is because a gets enabled when $x = 1$, and at the same time instant, it is reset to 0. Doing a twice, that is, executing path aa gives $x = 0, y = 2, z = 2$. Once we do aa, the action b is not feasible anymore since it requires $y = 1$. Therefore, in order to execute c, both a and b need to be executed at every time stamp $i \in \{1, \ldots, 100\}$ after which we have a valuation with $x = 0, y = 0, z = 100$. This is when c can be executed. From this example, we can see that the witness for reachability can have length exponential in the size of the input, if the constants are encoded in binary. It turns out that even with unary encoding, the witness can be exponential in the size of the input. This is because there are exponentially many possible orderings of the clocks and all of them may need to be visited in order to get a witness. For the PSPACE-hardness, a reduction from the problem of deciding whether a linear bounded automaton accepts a given string to the problem of reachability in timed automata is provided in (Theorem 4.17, [Alur and Dill 1994]). In [Courcoubetis and Yannakakis 1992], it was shown that reachability is PSPACE-complete even for automata with three clocks. When there is a single clock, the problem is NLOGSPACE-complete [Laroussinie et al. 2004]. For a long time, the complexity of two clock timed automata was open. This was settled in [Fearnley and Jurdzinski 2013; 2015] where it was shown that the problem remains PSPACE-complete.

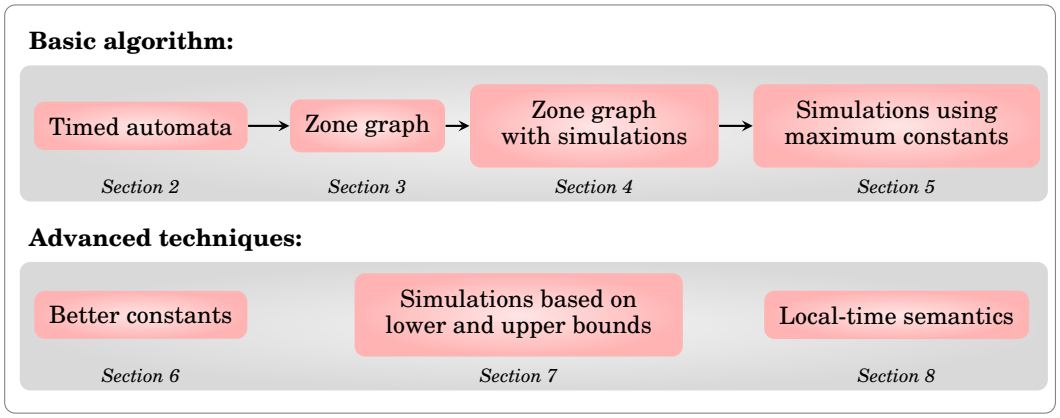
A naïve algorithm for reachability would be to enumerate paths in the automaton upto a sufficient bound and check whether one of them is feasible. For example, for automaton $\mathcal{A}_1$ of Figure 1, one could enumerate $c, ac, bc, aac, abc, \ldots$, etc. For enumerating paths, one could adopt some search order, for instance a breadth-first or a depth-first search. To check feasibility, it is convenient to store the set of clock values obtained after performing a particular path. For instance, to check feasibility of abc, if we know the set of clock values obtained after ab, we can then check whether this set intersects with the guard appearing in c. If it does, then abc is feasible. Else, it is not. The key feature in timed automata verification is the fact that this set - the set of clock values reached after executing a path - can be represented using a simple system of constraints called *zones*. These zones are amenable to efficient manipulation [Dill 1989]. Therefore checking feasibility is a non-issue. The bulk of the challenge is in figuring out how to do the enumeration sensibly, in particular, when to stop the enumeration. This is the main focus of this article.

The use of zones for timed automata reachability was advocated in [Daws and Tripakis 1998] and implemented in the tool KRONOS [Daws et al. 1995]. For stopping the enumeration, an extrapolation operation on zones was used: each new zone obtained

was inflated to a bigger zone. The number of such inflated zones was by definition finite, and hence whenever an already existing zone appears again, the enumeration from that path is stopped. A seminal work [Bouyer 2003; 2004] showed that this extrapolation operation was not correct if the guards of the timed automaton contain diagonal constraints, that is constraints of the form $x - y \leq 5, y - z \geq 2$, etc. Moreover, it was shown that no extrapolation can work when diagonal constraints are present. After this powerful result, the majority of work has concentrated on the fragment of timed automata without diagonal constraints. Several optimizations to the extrapolation approach have been studied, the most notable being [Behrmann et al. 2003; Behrmann et al. 2004; 2006]. These works are the default options in the tool UPPAAL [Larsen et al. 1997], which won the CAV award in 2013 for being the "foremost tool suite for the automated analysis and verification of real-time systems"[2].

In the last decade, a new approach to the zone enumeration has been investigated and implemented in an open-source tool TChecker [3]. In this approach, exploration of a path is stopped if the zone reached is *simulated* by an already existing zone obtained via a different path. This idea and the associated technical machinery were first proposed in [Herbreteau et al. 2011] and later refined in [Herbreteau et al. 2012; 2016]. This approach allows for a new bunch of optimizations [Herbreteau et al. 2013; Govind et al. 2019] and can also be extended to automata with diagonal constraints and updates (which are generalizations of the reset operation to allow assignments like $x := y + 2, x := x - 1$, etc to clocks in the transitions) [Gastin et al. 2018; 2019; 2020].

In this article, we will describe the simulation approach and some of the optimization techniques. The plan of the document is depicted below. Sections 2 to 5 give the basic zone based algorithm. The next three sections discuss some advanced optimizations. They are orthogonal to each other and can be read independently. Section 9 talks about the open-source tool TChecker that implements the algorithms discussed in the article. Finally in Section 10 we give some conclusions and perspectives.



PLAN OF THE DOCUMENT

[2]http://i-cav.org/cav-award/
[3]https://github.com/fredher/tchecker

2. TIMED AUTOMATA AND THE REACHABILITY PROBLEM

We will write $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}_{\geq 0}$ for the set of natural numbers, integers and non-negative reals, respectively. We will write 2^S for the power set of a set S.

A *clock* is a variable over $\mathbb{R}_{\geq 0}$. Fix a finite set X of clocks for the rest of the document. The set of *guards* $\Phi(X)$ is obtained using the grammar:

$$\phi := x \sim c \mid \phi \wedge \phi$$

where $x \in X$, $\sim \in \{\leq, <, >, \geq\}$, and $c \in \mathbb{N}$. The base constraints $x \sim c$ in the above grammar will be called *atomic constraints*. We will use a short form $x = c$ for the guard obtained by the conjunction of atomic constraints $x \leq c$ and $x \geq c$.

A *valuation* $v : X \to \mathbb{R}_{\geq 0}^{|X|}$ is a function assigning a non-negative real to each clock. Valuations will be the basic objects that will be used for the analysis of timed automata. Here are some notions on valuations. Let v be a valuation.

— *Time elapse:* for $\delta \in \mathbb{R}_{\geq 0}$, we write $v+\delta$ for the valuation such that $(v+\delta)(x) = v(x)+\delta$ for all x,
— *Guard satisfaction:* for a guard $g \in \Phi(X)$, we write $v \models g$ if $v(x) \sim c$ for each atomic constraint $x \sim c$ in g,
— *Reset:* for $R \subseteq X$, we write $v[R]$ for the valuation such that $v[R](x) = 0$ for $x \in R$ and $v[R](x) = v(x)$ for $x \notin R$,
— *Initial:* we write $\mathbf{0}$ for the valuation that maps every clock to 0. It will be called the initial valuation.

Definition 2.1 (*Timed automata [Alur and Dill 1994]*). A timed automaton $\mathcal{A} = (Q, q_0, \Sigma, X, T, F)$ is given by a finite set of states Q, an initial state $q_0 \in Q$, a finite alphabet Σ, a set of transitions $T \subseteq Q \times \Sigma \times \Phi(X) \times 2^X \times Q$ and a finite set $F \subseteq Q$ of accepting states. Each transition $(q, a, g, R, q') \in T$ has a source state q, target state q', a letter a, a guard g and a set of clocks R that are reset in the transition.

A *configuration* of a timed automaton is a pair (q, v) consisting of a state $q \in Q$ and a valuation v.

Definition 2.2 (*Semantics of a timed automaton*). The semantics of a timed automaton $\mathcal{A}$ is a transition system $\mathcal{S}_\mathcal{A}$ whose nodes consist of the set of all configurations of $\mathcal{A}$. The initial node is $(q_0, \mathbf{0})$ where q_0 is the initial state of $\mathcal{A}$ and $\mathbf{0}$ the initial valuation. The transition relation $\to$ is a union of two kinds of transitions:

delay. $(q, v) \to^\delta (q, v + \delta)$ for all $\delta \in \mathbb{R}_{\geq 0}$
action. $(q, v) \to^t (q', v')$ if $t = (q, a, g, R, q') \in T$ is a transition of $\mathcal{A}$, $v \models g$ and $v' = v[R]$.

We write $(q, v) \xrightarrow{\delta,t} (q_1, v_1)$ in short for the sequence $(q, v) \to^\delta \to^t (q_1, v_1)$ of delay δ followed by action t starting from (q, v).

A *run* of $\mathcal{A}$ is an alternating sequence of delay and action transitions in $\mathcal{S}_\mathcal{A}$ starting from the initial configuration: $(q_0, \mathbf{0}) \xrightarrow{\delta_0,t_0} (q_1, v_1) \xrightarrow{\delta_1,t_1} \cdots \xrightarrow{\delta_{n-1},t_{n-1}} (q_n, v_n)$. The run is *accepting* if $q_n \in F$.

Definition 2.3 (*Reachability problem*). The reachability problem for timed automata takes as input a timed automaton $\mathcal{A}$ and asks whether it has an accepting run.

We call the above problem as reachability and not emptiness since our focus is on algorithms, which are insensitive to the actual letters used. We still keep the alphabet

in our definition as it is convenient to explain the examples described in the document. We would essentially be using the alphabet to name the transitions.

3. ZONE GRAPH OF A TIMED AUTOMATON

In this section, we formalize the idea of enumerating paths of the automaton as discussed in Section 1.

3.1. Symbolic transition relation

We first define a transition relation $\Rightarrow$ over nodes of the form (q, W) where W is a set of valuations.

Definition 3.1 (*Symbolic transition relation* $\Rightarrow$). Let $\mathcal{A}$ be a timed automaton. For every transition $t = (q, a, g, R, q')$ of $\mathcal{A}$, we have a transition $\Rightarrow^t$ defined as follows:

$$(q, W) \Rightarrow^t (q', W') \text{ when } W' = \{v' \mid \exists v \in W,\ \exists \delta \in \mathbb{R}_{\geq 0}.\ (q, v) \rightarrow^t \rightarrow^\delta (q', v')\}$$

The transition relation $\Rightarrow$ is the union of all $\Rightarrow^t$.

The transition relation defined above considers each valuation $v \in W$ that can take the transition t, obtains the valuation v_t after performing the transition and then collects all time-successors of v_t. It is action followed by time, as opposed to the convention $\xrightarrow{\delta,t}$ that we used in the definition of runs. Therefore the symbolic transition $\Rightarrow$ always yields sets closed under time-successors. The initial configuration of the automaton is $(q_0, \mathbf{0})$. Starting from the initial valuation $\mathbf{0}$ the set of valuations reachable by a time elapse at the initial state is given by $\{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$. Call this W_0. From (q_0, W_0) as the initial node, computing the symbolic transition relation $\Rightarrow$ leads to different nodes (q, W) wherein the sets W are closed under time-successors.

Example 3.2. Consider the automaton with two clocks shown in Figure 2. The sets of valuations computed using Definition 3.1 are shown on the top of the automaton. The bright red part shows the valuations obtained immediately after the action, and the faded red part shows the ones obtained exclusively through time elapse. We explain some of these computations.

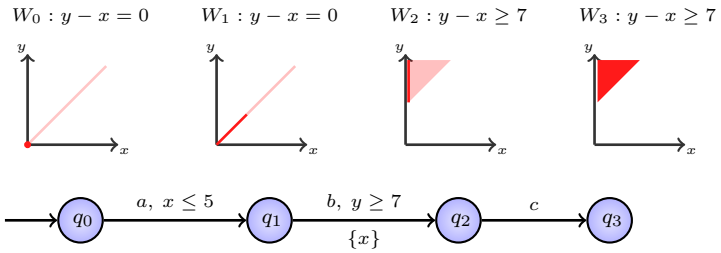


Fig. 2. Illustrating symbolic transitions. Trivial constraints like $x \geq 0$ and $y \geq 0$ are not written.

At the initial node, we start with the initial valuation $x = 0, y = 0$. Doing a time elapse δ gives $x = \delta, y = \delta$. Therefore, the set of all time successors of $x = 0, y = 0$ is given by $\{x = \delta, y = \delta \mid \delta \geq 0\}$. This is succinctly represented as $y - x = 0 \wedge x \geq 0$. Consider $(q_1, y - x = 0 \wedge x \geq 0)$ and the outgoing transition b. The action b has guard $y \geq 7$. The set of valuations that satisfy this guard is given by $y - x = 0 \wedge y \geq 7$, representing the valuations $\{x = \delta, y = \delta \mid \delta \geq 7\}$. Action b has a reset $\{x\}$. Resetting x gives the set $\{x = 0, y = \delta \mid \delta \geq 7\}$. This is depicted by the bright red line $x = 0 \wedge y \geq 7$ in the picture corresponding to W_2. The time elapse operation can be seen as picking

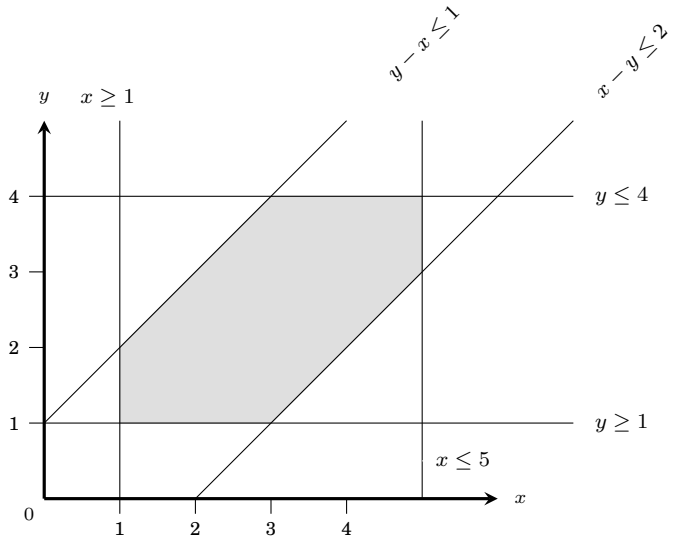


Fig. 3. An example of a zone

a point and then moving to the right parallel to the diagonal $x = y$. Doing the time elapse from each point in $\{x = 0, y = \delta \mid \delta \geq 7\}$ gives the set $y - x \geq 7 \wedge x \geq 0$: a point $x = \delta_1, y = \delta_2$ with $\delta_2 - \delta_1 \geq 7$ is obtained from $x = 0, y = \delta_2 - \delta_1$ by a time elapse δ_1.

3.2. Zones

As seen in the example, the sets computed are represented using simple constraints. This turns out to be a general property of timed automata. The sets W obtained in the nodes (q, W) computed using a sequence of symbolic transitions can be described by some simple constraints involving only the difference between clocks [Daws and Tripakis 1998; Bengtsson and Yi 2004]. This leads to the definition of *zones*.

Definition 3.3 (*Zones [Daws and Tripakis 1998]*). A zone is a set of valuations defined by a conjunction of two kinds of clock constraints: for $x, y \in X$

$$x \sim c$$
$$x - y \sim c$$

where, $\sim \in \{\leq, <, =, >, \geq\}$ and $c \in \mathbb{Z}$. For example, $(x > 4 \ \wedge \ y - x \leq 1)$ is a zone.

The sets depicted in Figure 2 are zones. Another example of a zone is illustrated in Figure 3. It can be shown that for transitions $(q, W) \Rightarrow (q', W')$, if W is a zone then so is W' [Daws and Tripakis 1998; Bengtsson and Yi 2004]. Observe that the initial set of valuations $W_0 = \{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$ is indeed a zone: it is given by the constraints

$$\bigwedge_{x,y \in X} (x \geq 0 \ \wedge \ x - y = 0)$$

Therefore every sequence of symbolic transitions leads to a zone. In the sequel, zones are denoted by Z, Z', etc.

Definition 3.4 (*Zone graph*). Given a timed automaton $\mathcal{A}$, the *zone graph* $ZG(\mathcal{A})$ of $\mathcal{A}$ is a transition system whose nodes are of the form (q, Z) with q a control state of $\mathcal{A}$

and Z a zone. The initial node is (q_0, Z_0) where $Z_0 = \{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$. The transitions are given by the relation $\Rightarrow$ of Definition 3.1.

As zones have a simple description, they can be efficiently represented using what are called Difference-Bound Matrices (DBMs) [Dill 1989]. The successor computation $(q, Z) \Rightarrow^t (q', Z')$ for a transition $t = (q, a, g, R, q')$ proceeds in the following steps.

$$(q, Z) \xrightarrow{guard} (q, Z \wedge g) \xrightarrow{reset} (q, (Z \wedge g)[R]) \xrightarrow{elapse} (q, \overrightarrow{(Z \wedge g)[R]})$$

where the operations on the zones are as defined:

$$Z \wedge g = \{v \mid v \in Z \text{ and } v \models g\}$$
$$Z_1[R] = \{v[R] \mid v \in Z_1\}$$
$$\overrightarrow{Z_2} = \{v + \delta \mid v \in Z_2, \delta \in \mathbb{R}_{\geq 0}\}$$

All these operations can be computed efficiently using DBMs. The costliest operation is the computation of the intersection of a zone with a guard. It has been shown in [Zhao et al. 2005] that the intersection operation can be done in $\mathcal{O}(|X|^2)$ time. The other operations are easier, and therefore the entire successor computation can be done in $\mathcal{O}(|X|^2)$, a complexity quadratic in the number of clocks. Extensions of timed automata have been defined where guards could contain diagonal constraints like $x - y \leq 4$. In this case, the intersection takes $\mathcal{O}(|X|^3)$.

3.3. A certificate for (un)reachability

To solve reachability, we want an algorithm that produces a certificate ascertaining whether a given state q is reachable or not in the timed automaton. A certificate for reachability is a run leading to state q. Certificates for unreachability are less obvious. We want an object that contains exactly the set of reachable control states, in some form. The zone graph acts as a certificate: if there is no node with state q in the zone graph, we can conclude that q is not reachable in the timed automaton. This is due to the completeness of the zone graph that we state below. The next theorem follows simply from the definition of the symbolic transition.

THEOREM 3.5. *The zone graph $ZG(\mathcal{A})$ of a timed automaton $\mathcal{A}$ satisfies the following properties:*

— Soundness. *for every path $(q_0, Z_0) \Rightarrow^{t_1} (q_1, Z_1) \Rightarrow^{t_2} \ldots \Rightarrow^{t_n} (q_n, Z_n)$ in $ZG(\mathcal{A})$ there is a run $(q_0, v_0) \xrightarrow{\delta_1, t_1} (q_1, v_1) \xrightarrow{\delta_2, t_2} \cdots \xrightarrow{\delta_n, t_n} (q_n, v_n)$ in $\mathcal{S}_\mathcal{A}$ such that $v_0 = \mathbf{0}$ and $v_i \in Z_i$ for all $0 \leq i \leq n$.*

— Completeness. *for every run $(q_0, v_0) \xrightarrow{\delta_1, t_1} (q_1, v_1) \xrightarrow{\delta_2, t_2} \cdots \xrightarrow{\delta_n, t_n} (q_n, v_n)$ with $v_0 = \mathbf{0}$ in $\mathcal{S}_\mathcal{A}$ there is a path $(q_0, Z_0) \Rightarrow^{t_1} (q_1, Z_1) \Rightarrow^{t_2} \ldots \Rightarrow^{t_n} (q_n, Z_n)$ in $ZG(\mathcal{A})$ with $v_i \in Z_i$ for all $0 \leq i \leq n$.*

The theorem suggests to compute the zone graph to solve reachability. However, there is one major hurdle: zone graphs can be infinite in general. Figure 4 shows an automaton $\mathcal{A}_2$ in the left, the zones obtained by executing paths a^n in the middle and the zone graph in the right. We do not explicitly mark any accept states in the automaton, since we are interested in looking at the entire zone graph which contains all the reachable control states. In the zone graph $ZG(\mathcal{A}_2)$, edges marked with a red cross are *disabled*, that is, there is no valuation in the zone that satisfies the guard of the transition corresponding to the edge. For example, the transition b is disabled from the zone obtained after aa. Hence the path aab is not feasible. As we notice in this example, the zone graph is infinite.

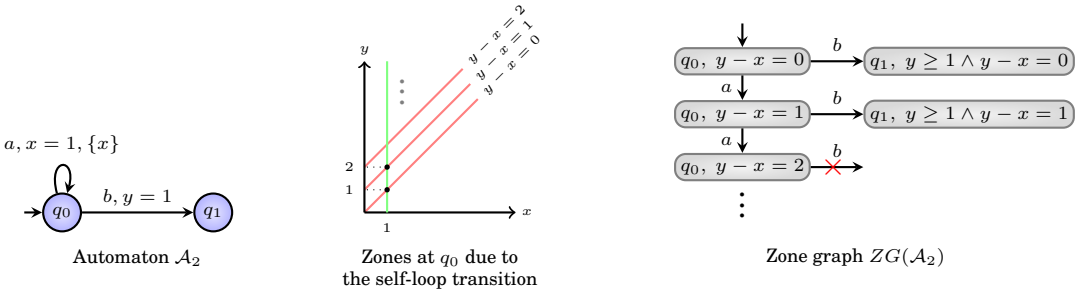


Fig. 4. Zone graph could be infinite

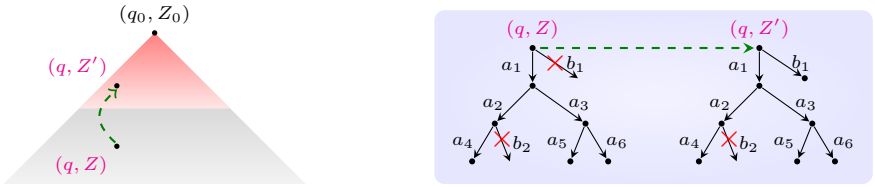


Fig. 5. Computing a finite prefix of the zone graph using simulations. When node (q, Z) is simulated by (q, Z'), all sequences feasible from (q, Z) are feasible from (q, Z') too.

A naïve exploration of the zone graph therefore does not work. Can we find a finite prefix of the zone graph that contains all reachable control states? That is, for every node (q, Z) outside this prefix, there exists a representative node (q, Z') inside the prefix, having the same control state. In addition to asking just for a finite prefix, one would like a prefix which is as small as possible and yet contains all reachable states. Moreover, the test for sufficiency of the prefix should be efficient. The subsequent sections provide an answer to this question.

4. PRUNING THE ZONE GRAPH USING SIMULATIONS

We will now discuss a mechanism to compute a finite prefix that contains all reachable states. The idea is depicted in Figure 5. On the left, the potentially infinite zone graph starting from the initial node (q_0, Z_0) is shown in gray. The red portion is the finite prefix that contains all reachable control states. For every node (q, Z) that is outside this prefix, there is a node (q, Z') inside the prefix such that (q, Z) is *simulated* by (q, Z'). Intuitively, if (q, Z) is simulated by (q, Z') every sequence w of actions feasible from (q, Z) is feasible from the bigger node (q, Z'). This is illustrated in Figure 5 by the picture on the right. Therefore it is possible to explore w from (q, Z') while staying inside the prefix. This also ensures that every reachable control state is present inside the prefix.

The main challenge now is to come up with a concrete simulation relation which can be applied on zones. A simple solution would be to say (q, Z) is simulated by (q, Z') whenever $Z \subseteq Z'$. It is easy to see that this is correct: whatever can be done from (q, Z) can also be done from (q, Z'). However, applying this operation in the example of Figure 4 does not give a finite prefix. Notice that the zones appearing at q_0 are all incomparable and hence a mere inclusion is useless. One needs a more sophisticated simulation. We will first formalize the concept of simulations and present a concrete simulation relation in Section 5.

4.1. Simulations

We will define a relation over the space of configurations that satisfies some "one-step" properties. This will then be lifted to zones.

Definition 4.1 (*Simulation*). A simulation for a timed automaton $\mathcal{A}$ is a reflexive and transitive relation $\preceq$ on its semantics $\mathcal{S}_\mathcal{A}$ that relates configurations having the same control state, and satisfies two other properties. For all $(q, v) \preceq (q, v')$, we have:

(1) $(q, v + \delta) \preceq (q, v' + \delta)$ for every $\delta \in \mathbb{R}_{\geq 0}$, and
(2) for every transition $t = (q, a, g, R, q_1)$, if $(q, v) \to^t (q_1, v_1)$ then $(q, v') \to^t (q_1, v_1')$ and $(q_1, v_1) \preceq (q_1, v_1')$.

We extend the definition to zones, by saying $(q, Z) \preceq (q, Z')$ if for every $v \in Z$ there exists $v' \in Z'$ such that $(q, v) \preceq (q, v')$.

The above definition is sometimes called a strong-timed simulation in the literature [Tripakis and Yovine 2001]. In the first condition above, we say that for every time elapse δ, we have $(q, v + \delta) \preceq (q, v' + \delta)$. Instead we could relax the condition by saying for every $\delta \in \mathbb{R}_{\geq 0}$ there exists $\delta' \in \mathbb{R}_{\geq 0}$ such that $(q, v + \delta) \preceq (q, v' + \delta')$. This is enough since we are only interested in control state reachability. We do not care about the exact times taken while reaching a state. This relaxed definition asking for an arbitrary δ' is known as a time-abstract simulation. However, the concrete simulations that we know all turn out to be strong-timed simulations. Hence we stick to Definition 4.1. The next lemma follows directly from Definitions 3.1 and 4.1.

LEMMA 4.2. *Let $\preceq$ be a simulation for a timed automaton $\mathcal{A}$. Let $(q, Z) \preceq (q, Z')$. For every sequence of transitions $(q, Z) \Rightarrow^{t_1} (q_1, Z_1) \Rightarrow^{t_2} \cdots \Rightarrow^{t_n} (q_n, Z_n)$ in $ZG(\mathcal{A})$ there exists a sequence $(q, Z') \Rightarrow^{t_1} (q_1, Z_1') \Rightarrow^{t_2} \cdots \Rightarrow^{t_n} (q_n, Z_n)$ in $ZG(\mathcal{A})$ such that $(q_i, Z_i) \preceq (q_i, Z_i')$.*

The above lemma guarantees that stopping the zone graph computation at (q, Z) and continuing it from (q, Z') is sufficient for control state reachability. The next important question is that of finiteness. What kind of simulations can generate a finite prefix?

Definition 4.3. A simulation $\preceq$ is said to be finite if for every state q and for every sequence $Z_1, Z_2, \ldots$ of zones there exist indices i, j with $j > i$ such that $(q, Z_j) \preceq (q, Z_i)$.

When a finite simulation is employed, there can be no infinite paths explored. At some point we will hit a node (q, Z_j) that is simulated by an existing node (q, Z_i). Finite simulations therefore induce finite prefixes of the zone graph.

4.2. Simulation graphs and the reachability algorithm

We will call the prefixes induced by simulations as simulation graphs. When the simulation is finite, we can get a finite simulation graph.

Definition 4.4 (*Simulation graph*). Let $\mathcal{A}$ be a timed automaton and $\preceq$ a simulation relation. A simulation graph $ZG^{\preceq}(\mathcal{A})$ is a subset of nodes and edges of the zone graph $ZG(\mathcal{A})$ along with some new edges called *simulation edges*. Each node is marked either *uncovered* or *covered*. The simulation graph satisfies the following conditions.

— The initial node (q_0, Z_0) is present in $ZG^{\preceq}(\mathcal{A})$ and is marked uncovered.
— For every uncovered node (q, Z), all its successors along with the associated transitions are present in $ZG^{\preceq}(\mathcal{A})$: that is, every edge $(q, Z) \Rightarrow^t (q_1, Z_1)$ in $ZG(\mathcal{A})$ is also present in the simulation graph.
— For every covered node (q, Z), there exists an uncovered node (q, Z') such that $(q, Z) \preceq (q, Z')$. In such a case, there is a simulation edge $(q, Z) \dashrightarrow (q, Z')$.

The above conditions do not necessarily give a unique graph. We denote by $ZG^{\preceq}(\mathcal{A})$ some arbitrarily picked simulation graph for $\mathcal{A}$.

The idea is that from covered nodes we do not explore further. The set of uncovered nodes forms the finite prefix and the covered nodes are the border between the prefix and the rest of the zone graph. Let us introduce one more notation before we discuss the correctness of simulation graphs. We want to capture the combination $\Rightarrow^t \dashrightarrow$ consisting of a successor edge followed by a simulation edge, whenever such a sequence exists. Let (q, Z) be an uncovered node and t an outgoing transition from q. We write $(q, Z) \rightsquigarrow^t (q_1, Z_1')$ when $(q, Z) \Rightarrow^t (q_1, Z_1)$ and:

— either (q_1, Z_1) is uncovered and $Z_1' = Z_1$,
— or (q_1, Z_1) is covered and $(q_1, Z_1) \dashrightarrow (q_1, Z_1')$.

Notice that (q_1, Z_1') will be an uncovered node. The next theorem gives an analogue of Theorem 3.5 for simulation graphs.

THEOREM 4.5. *The simulation graph $ZG^{\preceq}(\mathcal{A})$ satisfies the following two properties:*

— Soundness: *for every $(q_0, Z_0) \Rightarrow^{t_1} (q_1, Z_1) \Rightarrow^{t_2} \cdots \Rightarrow^{t_n} (q_n, Z_n)$ in $ZG^{\preceq}(\mathcal{A})$ there is a run $(q_0, v_0) \xrightarrow{\delta_1, t_1} (q_1, v_1) \cdots \xrightarrow{\delta_n, t_n} (q_n, v_n)$ in $\mathcal{S}_{\mathcal{A}}$ such that $v_0 = \mathbf{0}$ and $v_i \in Z_i$ for all $0 \leq i \leq n$.*
— Completeness: *for every run $(q_0, v_0) \xrightarrow{\delta_1, t_1} (q_1, v_1) \xrightarrow{\delta_2, t_2} \cdots \xrightarrow{\delta_n, t_n} (q_n, v_n)$ of $\mathcal{S}_{\mathcal{A}}$ there exists a path $(q_0, Z_0) \rightsquigarrow^{t_1} (q_1, Z_1') \rightsquigarrow^{t_2} \cdots \rightsquigarrow^{t_n} (q_n, Z_n')$ in $ZG^{\preceq}(\mathcal{A})$ such that for every v_i there exists $v_i' \in Z_i'$ with $(q_i, v_i) \preceq (q_i, v_i')$.*

Observe that the difference between Theorems 3.5 and 4.5 occurs in the completeness condition. For zone graphs, every run of $\mathcal{A}$ had a representative sequence of zones in the zone graph with valuation v_i by itself being present in zone Z_i. In the simulation case, every v_i has a bigger v_i' in the corresponding zone Z_i'.

We can now describe a reachability algorithm that computes a simulation graph. The algorithm is just a pseudocode-like summary of the above discussion. When the simulation is finite, the algorithm terminates with a finite simulation graph. Correctness of the algorithm simply follows from Theorem 4.5.

Reachability algorithm. Input is a timed automaton $\mathcal{A}$, a set of accept states F and a finite simulation $\preceq$. Algorithm uses two lists $Passed$ and $Waiting$.

(1) Initialization: If $q_0 \in F$, return Yes. Else, add the initial node (q_0, Z_0) to $Waiting$.
(2) While $Waiting$ is non-empty. Pick and remove a node (q, Z) from $Waiting$. Add it to $Passed$. For all transitions $t = (q, a, g, R, q_1)$, compute $(q, Z) \Rightarrow^t (q_1, Z_1)$.
    — If $q_1 \in F$, return Yes.
    — Else, if there exists a node (q_1, Z_1') in $Passed$ or $Waiting$ such that $(q_1, Z_1) \preceq (q_1, Z_1')$, discard (q_1, Z_1).
    — Else add (q_1, Z_1) to Waiting.
(3) Return No (when $Waiting$ becomes empty and algorithm has not stopped at (2)).

4.3. Where is the challenge?

So far, in this section, we have discussed a framework using simulations to get a reachability algorithm. But one needs to instantiate this framework with a concrete simulation relation. Moreover, notice that the test $(q, Z) \preceq (q, Z')$ is used frequently in the reachability algorithm: each fresh node is simulation-tested with potentially many existing nodes. Therefore, we require a simulation relation for which this test is efficient. In zone technology, it is difficult to do better than $\mathcal{O}(|X|^2)$, a quadratic complexity in

the number of clocks. Successor computation and inclusion between zones have this complexity. In some sense, we need to go through every constraint in the zone, and there is a constraint for every pair of clocks. So it is reasonable to assume we cannot do better than $\mathcal{O}(|X|^2)$. Therefore, our aim would be to construct a simulation relation that is finite and for which the simulation test can be done in $\mathcal{O}(|X|^2)$.

5. SIMULATION BASED ON MAXIMUM CONSTANTS

It is clear from Example 4 that a simple inclusion $Z \subseteq Z'$ does not induce a finite simulation. We need a way to relate valuations with different values for clocks. In the paper introducing timed automata [Alur and Dill 1994], a fundamental observation was made, which has been reused several times in the literature: once the value of a clock goes beyond the maximum constant appearing among the guards of the automaton, its actual value does not matter. With this in mind, we introduce a simulation relation that is parameterized by a *bounds function* associating a constant to every clock. This relation was proposed in [Behrmann et al. 2006].

Definition 5.1 (M*-bounds*). An M-bound is a function $M : X \to \mathbb{N} \cup \{-\infty\}$ that maps each clock to a natural number or $-\infty$. A constraint $x \sim c$ conforms to M if $c \leq M(x)$. A guard g conforms to M if every atomic constraint in g conforms to M. A timed automaton $\mathcal{A}$ conforms to M if every constraint $x \sim c$ appearing in a guard of $\mathcal{A}$ conforms to M.

Definition 5.2 (M*-equivalence*). The M-equivalence $\equiv_M$ is a relation between valuations that is parameterized by an M-bound. We say $v \equiv_M v'$ if for all clocks x:

$$\text{either } v(x) = v'(x) \quad \text{or} \quad \text{both } v(x) > M(x) \text{ and } v'(x) > M(x)$$

We extend the relation to configurations by saying $(q, v) \equiv_M (q, v')$ if $v \equiv_M v'$.

Example 5.3. Consider automaton $\mathcal{A}_2$ in Figure 4. Here is an M-bound that $\mathcal{A}_2$ conforms to: $M(x) = 1, M(y) = 1$. Consider three valuations: $v_1 : x = 0, y = 1, v_2 : x = 0, y = 2, v_3 : x = 0, y = 3$. Notice that $v_2 \equiv_M v_3$, but $v_1 \not\equiv_M v_2$ since $v_1(y) \leq M(y)$ and $v_2(y) > M(y)$. Valuation v_1 satisfies the guard $y = 1$ whereas v_2 does not satisfy the same guard. On the other hand v_2 and v_3 satisfy the same set of guards that conform to M.

LEMMA 5.4. *Let $v \equiv_M v'$ for a given M-bound, and let $x \sim c$ be a constraint that conforms to M. Then, $v \models (x \sim c)$ iff $v' \models (x \sim c)$.*

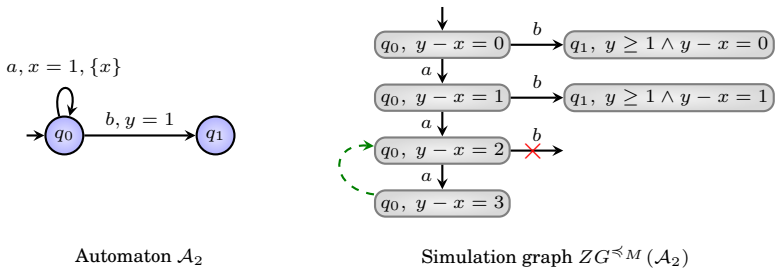
It is easy to see that $\equiv_M$ is an equivalence, and so we have already called it the M-equivalence. What we require is a simulation. The next theorem states that $\equiv_M$ is in fact a bisimulation for automata that conform to M. Proof of the theorem follows by verifying that $\equiv$ satisfies the conditions mentioned in Definition 4.1 that are required for a simulation. Condition (2) follows thanks to Lemma 5.4.

THEOREM 5.5. *Let $\mathcal{A}$ be a timed automaton that conforms to an M-bound. The relation $(q, v) \equiv_M (q, v')$ is a bisimulation for $\mathcal{A}$.*

We can extend $\equiv_M$ to a simulation on zones as in Definition 4.1: $(q, Z) \preccurlyeq_M (q, Z')$ if for all $v \in Z$ there exists $v' \in Z'$ such that $v \equiv v'$. Observe that $(q, Z) \preccurlyeq_M (q, Z')$ does not imply $(q, Z') \preccurlyeq_M (q, Z)$. Let $ZG^{\preccurlyeq_M}(\mathcal{A})$ be the the simulation graph obtained using this simulation $\preccurlyeq_M$.

Example 5.6. Figure 6 shows the simulation graph using $\preccurlyeq_M$ for the automaton $\mathcal{A}_2$, with the bounds function $M(x) = 1, M(y) = 1$.

The uncovered nodes are the ones reached by sequences ϵ, a, aa, b, ab. The action b is disabled from the node aa. The only covered node is aaa, which is covered by aa. The



Automaton $\mathcal{A}_2$ Simulation graph $ZG^{\preccurlyeq_M}(\mathcal{A}_2)$

Fig. 6. Automaton $\mathcal{A}_2$ from Figure 4 and its simulation graph using the $\preccurlyeq_M$ relation over the bounds $M(x) = 1, M(y) = 1$

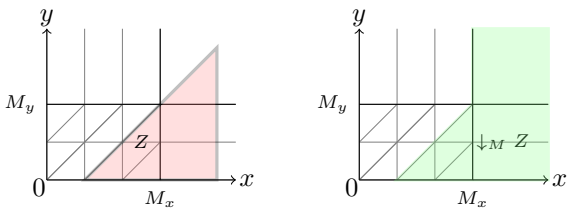


Fig. 7. A zone Z and its downward closure $\downarrow_M Z$. The thin grey lines give the division into regions with respect to M [Alur and Dill 1994]

zone reached after aaa is $y - x = 3$ and the one reached after aa is $y - x = 2$. Each valuation in aaa is of the form $v : x = i, y = i + 3$ where $i \geq 0$. This valuation v can be simulated by $v' : x = i, y = i + 2$ which is present in $y - x = 2$. This helps prune the graph at $y - x = 3$ resulting in a finite graph. Let us also see why we could not have pruned earlier. The zone $y - x = 2$ is not simulated by $y - x = 1$ because we have $v_2 : x = 0, y = 2$ in $y - x = 2$. The only valuation with $x = 0$ in $y - x = 1$ is $v_1 : x = 0, y = 1$. But $v_1 \not\equiv_M v_2$, as already seen in Example 5.3. For similar reasons, $y - x = 2$ is not covered by $y - x = 0$, and $y - x = 1$ is not covered by $y - x = 0$.

5.1. Finiteness

The relation $\preccurlyeq_M$ satisfies the basic requirement of being a simulation. What about finiteness? Notice that the equivalence $\equiv_M$ does not induce finitely many equivalence classes since for values less than the bound, we ask for exact correspondence $v(x) = v'(x)$. Nevertheless, we can prove that the simulation $\preccurlyeq_M$ over zones is finite as per Definition 4.3. Suppose we write $\downarrow_M Z = \{v \mid \exists v' \in Z \text{ s.t. } v \equiv_M v'\}$ for the downward closure of Z with respect to $\equiv_M$. We call it a "downward" closure because in general for a simulation $\preccurlyeq$, we want the set $\{v \mid \exists v' \in Z \text{ s.t. } v \preccurlyeq v'\}$, which is the set of all valuations that can be simulated by Z. For this specific case of $\equiv_M$ which is in fact a bisimulation, both the upward and downward closures are identical.

Coming back to the question of finiteness of $\preccurlyeq_M$, it can be shown that the downsets $\downarrow_M Z$ are unions of classical regions as defined in [Alur and Dill 1994]. This is illustrated in Figure 7.

The proof of this result is non-trivial. It follows from two observations.

— As the constraints in guards use natural numbers, the zones obtained during the algorithm are also defined using integer constants. Therefore, a region which has all clocks bounded is either entirely inside the zone or entirely outside it. For example, in Figure 7, region $2 < x < 3 \wedge 0 < y < 1 \wedge x > y$ is completely inside the zone Z, whereas the region $0 < x < 1 \wedge 1 < y < 2 \wedge x > y$ is completely outside.

— Suppose $v \equiv_M v'$. Let v_1 be region equivalent to v. Then there exists a v'_1 in the *neighbourhood* of v' such that $v'_1 \equiv_M v_1$. By neighbourhood of v', we mean the smallest zone (using integer constants) that contains v'. Therefore, any zone Z containing v' also contains its neighbourhood, and so in particular it will contain v'_1.

From the second point, we can derive that the downset $\downarrow_M Z$ is a union of regions: if $v \in \downarrow_M Z$, then $v_1 \in \downarrow_M Z$. The second observation above is called the adjustment lemma (Lemma 18 of [Herbreteau et al. 2016]) and appears in a generalized context. We state the finiteness of $\preccurlyeq_M$ in the following theorem.

THEOREM 5.7. *The simulation $\preccurlyeq_M$ is finite.*

5.2. Efficient simulation test

The final hurdle is the test $(q, Z) \preccurlyeq_M (q, Z')$. Notice that it does not really depend on the state q, and so we will just write $Z \preccurlyeq_M Z'$ in this section. We ask what is a witness for $Z \not\preccurlyeq_M Z'$? A witness for non-simulation is a valuation $v \in Z$ such that its equivalence class with respect to $\equiv_M$ does not intersect Z'. This is illustrated in Figure 8.

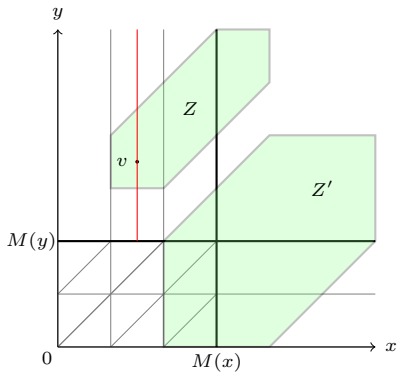


Fig. 8. We have zone $Z \not\preccurlyeq_M Z'$ since for $v \in Z$ there is no equivalent valuation in Z'. The red line depicts all valuations that are $\equiv_M$ equivalent to v.

When there are at most two clocks, it seems possible to check this by looking at the constraints of Z and Z'. When there are more than two clocks, it is difficult to visualize the witness. Quite remarkably, it turns out that if there is a witness v for $Z \not\preccurlyeq_M Z'$, then there are two clocks x, y such that the projection of v (call it $v(x, y)$) is a witness for $Z(x, y) \not\preccurlyeq_M Z'(x, y)$ where $Z(x, y)$ and $Z'(x, y)$ denote the projections of Z, Z' onto x, y. This means, in order to find a witness v for $Z \not\preccurlyeq Z'$, it is sufficient to run $Z(x, y) \not\preccurlyeq Z'(x, y)$ through all projections to two clocks. This immediately gives a complexity $\mathcal{O}(|X|^2)$. The proof of this result and the final simulation test can be found in Section 5 of [Herbreteau et al. 2016].

Thanks to Theorems 5.5 and 5.7, and the efficient inclusion test, we now have an algorithm for reachability that can be implemented efficiently: given automaton $\mathcal{A}$, take M to be the function setting $M(x)$ to the maximum constant that appears in $\mathcal{A}$, and $M(x) = -\infty$ if there is no constraint on x in $\mathcal{A}$, and then run the reachability algorithm with $\preccurlyeq_M$ to compute $ZG^{\preccurlyeq_M}(\mathcal{A})$.

6. COMPUTING BETTER CONSTANTS

Now that we have a simulation that can be put to work, what more do we want? A natural goal is to ask for simulations that can generate smaller simulation graphs. For this to materialize we require more simulations happening during the computation. If we stick to M-bounds as parameters, we cannot do better in terms of the simulation relation: the relation $\equiv_M$ is the biggest relation that is correct for all automata conforming to M [Herbreteau et al. 2012; 2016]. Therefore, we either need to extract more information about the automaton, or look for better constants M, that are still sufficient for the given automaton.

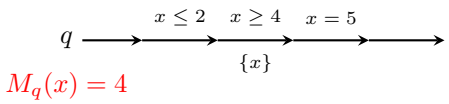
6.1. Smaller the bounds, better the simulation

Suppose for two bounds M_1 and M_2, we say $M_1 \leq M_2$ if $M_1(x) \leq M_2(x)$ for all clocks x. Notice that M_1 induces a coarser equivalence than M_2: if $v_1 \equiv_{M_2} v_2$, then $v_1 \equiv_{M_1} v_2$. Therefore, if $Z \preccurlyeq_{M_2} Z'$ we also have $Z \preccurlyeq_{M_1} Z'$, but not the other way around. Due to this, we can expect more simulations to happen when the bounds are smaller.

Theorem 5.5 suggests that any M-bound that the automaton conforms to is a good candidate. Just taking the maximum constant in $\mathcal{A}$ for each clock is a correct, but an overly conservative approach. We will describe three approaches to get better M-bounds.

6.2. State specific bounds via a static analysis

Consider the picture on the right in Figure 5 once again. Both nodes have the same control state q. The first observation is that for q, it is enough to look at paths starting from q [Behrmann et al. 2003]. A static analysis of the automaton is performed as a pre-processing step to compute bounds M_q for each state q of the automaton. At state q, the function $M_q(x)$ gives the largest constant appearing in a path starting from q which does not reset x. This is illustrated below. There is a path from q that leads to a constraint $x \geq 4$, without resetting x in between. Hence $M_q(x) = 4$ (assuming there are no other paths from q). The constant 5 is irrelevant at q due to the reset of x happening in between q and the constraint $x = 5$. Simulations involving state q can make use of the bound M_q: the simulation check becomes $(q, Z) \preccurlyeq_{M_q} (q, Z')$.



Consider automaton $\mathcal{A}_3$ in Figure 9. The bound M_{q_0} associates $-\infty$ to y since the path from q_0 to the constraint $y = 10^6$ has a reset of y. This gives three uncovered nodes in the simulation graph corresponding to paths ϵ, b, bc. The node $(q_0, Z_a) := (q_0, y - x = 1)$ reached after action a gets covered by $(q_0, Z_\epsilon) := (q_0, y - x = 0)$, the node corresponding to ϵ. Notice that a valuation $v := (x = i, y = i + 1) \in Z_a$ is simulated by $v' := (x = i, y = i) \in Z_\epsilon$ as $M_{q_0}(y) = -\infty$. A global bound which is not state-specific associates 10^6 to y, resulting in at least 10^6 nodes at q_0.

6.3. Bounds for each (q, Z) by an on-the-fly method

The second observation is that for node (q, Z), it is in fact sufficient to consider only the part of the zone graph below (q, Z). Hence there is a bound function $M_{(q,Z)}$ for each node (q, Z). This is in principle better since there could be a large constant that appears to be possible from q, but one may never reach it in any run of the timed automaton. This is illustrated in automaton $\mathcal{A}_4$ in Figure 9. Action b is not feasible, no

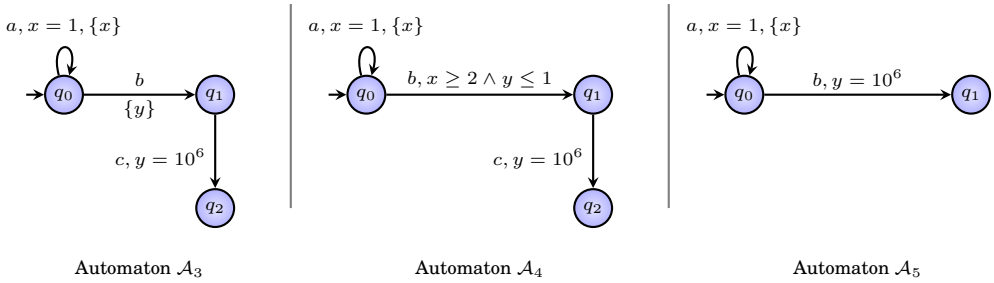


Fig. 9. Automata examples to illustrate better bounds computation.

matter how many times we do the loop a. Therefore the constraint $y = 10^6$ with the big constant is never reachable in the zone graph. The state specific bounds computation via static analysis does not recognize this. It would assign a bounds function M_{q_0} at q_0 with $M_{q_0}(y) = 10^6$. This would result in a simulation graph with at least 10^6 nodes.

When we first visit (q, Z), we do not have the subgraph below it and so we do not have $M_{(q,Z)}$ at that point. In [Herbreteau et al. 2011], an algorithm has been proposed that incorporates the bounds computation along with the zone graph exploration. Initially all bounds are assumed to be $-\infty$ and they are increased as and when new constraints are seen during the exploration. From each new transition that is seen, the constants are propagated backwards along the predecessors until all resets are hit, or the initial node is reached. For the static analysis bounds we had an M_q and it was clear that the simulation $(q, Z) \preccurlyeq_{M_q} (q, Z')$ would use M_q. Now, when we assign an $M_{(q,Z)}$ for each (q, Z) which bounds do we use for the simulation between (q, Z) and (q, Z'): the bound function $M_{(q,Z)}$ or $M_{(q,Z')}$? Recall that when we do a simulation $(q, Z) \dashrightarrow (q, Z')$, the node (q, Z) is covered and (q, Z') is uncovered. The idea of this simulation edge is that: based on whatever has been seen from (q, Z'), the node (q, Z) is covered and exploring (q, Z) will not see anything that is unseen from (q, Z'). Therefore for the simulation check, the bound $M_{(q,Z')}$ are used. The more we explore, the more we learn about the subgraph from (q, Z'). Hence these bounds are dynamically changing and the coverings may need to be revisited and removed. So the algorithm from [Herbreteau et al. 2011] is more sophisticated than a plain enumeration of zones. It goes through an exploration phase, followed by a refinement phase to check for bad coverings. The process is repeated until there are no bad coverings.

6.4. Bounds for each (q, Z) by a lazy approach

One could go a step further. We look at Figure 5 again. Let us now look at the situation from the point of view of node (q, Z'). Suppose there is no disabled edge in the part of the zone graph below (q, Z'). This means that every sequence that is possible from q in the automaton has been seen from (q, Z'). Such a node can simulate any other node that contains the same control state q. The on-the-fly computation of Section 6.3 would take into consideration all the constants in the subgraph starting from (q, Z'). When there is a non-trivial M-bound, node (q, Z') may not simulate every other reachable node. The best option in such a situation is to associate a trivial M-bound that gives $-\infty$ to every clock.

Consider automaton $\mathcal{A}_5$ from Figure 9. If the zone graph is computed, one would notice that b is feasible from $(q_0, y - x = 0)$. Now, on doing a, we reach $(q_0, y - x = 1)$. It is ok to cover $(q_0, y - x = 1)$ by $(q_0, y - x = 0)$. This is because there are no disabled edges from $(q_0, y - x = 0)$ and hence every control state reachable from q_0 has been seen. There is no point in further exploration from $(q_0, y - x = 1)$. When the bounds

are trivial at $(q_0, y - x = 0)$, this covering will indeed happen. So for this automaton, the algorithm starts with trivial bounds and does not increase them any further.

This was an exceptional situation. What about when there is some disabled edge a from (q, Z')? Node (q, Z') cannot simulate node (q, Z) if the latter can do action a. To let the algorithm know this, we need to pick the guard in a and incorporate it into the M-bound for (q, Z'). Now, what about a disabled edge that comes later, say there is a sequence w from (q, Z') leading to a node (q_w, Z'_w) from which action a is disabled.

We need to construct a correct M-bound at (q, Z') that ensures that (q, Z') does not simulate nodes from which wa is feasible. This is achieved as follows in [Herbreteau et al. 2013]:

(1) Add constants of a to $M_{(q_w, Z'_w)}$.
(2) Propagate these constants backward along w to increase $M_{(q,Z')}$, picking only a subset of constraints in w that cause a to be disabled.

There could be various reasons why a is disabled at (q_w, Z'_w): for instance a is $x \leq 3$ and there is a guard $x \geq 5$ in w after which there is no reset. There could be yet another guard $y \geq 4$ in w, but we have the constraint $x = y$ in the zone just before it. This generates a derived constraint $x \geq 4$ which could have caused a to be disabled. The algorithm from [Herbreteau et al. 2013] makes one choice of guards to pick and shows them to be sufficient. The bounds computed this way are called lazy M-bounds. Once again, this lazy approach makes use of an intricate algorithm that starts with all bounds being $-\infty$ and dynamically updates them during the exploration by back-propagating guards starting from a disabled edge. An example exhibiting exponential gains with this approach is presented in [Herbreteau et al. 2013].

In summary, the lazy approach triggers the back-propagation of bounds only from disabled edges, and moreover the propagation algorithm makes an intelligent choice of constraints to pick.

6.5. Which bounds are the best?

There is a trade-off between the static bounds computation and the dynamic bounds computation: the computed bounds are smaller in the dynamic case, but there is a computation overhead involved during the algorithm. The dynamic computed bounds may prune out a big portion of the state space in which case the performance (in terms of memory consumption and time) is better compared to the algorithm with the static analysis bounds. If this does not happen, there is no gain due to dynamic bounds and in fact the extra computation results in a longer time to completion. There are examples to show both situations. The default option in tools is the static analysis method for deriving constants which seems to work well for most of the examples. The general sentiment seems to be that keeping the algorithm simple gives good performance. But, we do think that it is an interesting direction to look for better algorithms for the dynamic bounds computation and perform extensive experimentation to compare the benefits.

7. DISTINGUISHING LOWER AND UPPER BOUNDS

In Section 5, we have defined a simulation $\equiv_M$ on valuations and a relation $\preccurlyeq_M$ on zones which leads to an associated simulation graph $ZG^{\preccurlyeq_M}(\mathcal{A})$. In the subsequent Section 6, we have seen methods to get smaller M-bounds. In this section, we will

present better parameters for the simulation by distinguishing lower bound and upper bound constraints occurring in the guards.

7.1. The key idea

Suppose for a state q, the set of relevant guards seen by the static analysis method is $\{x \leq 2, y \geq 3\}$. The M-bound would associate $M_q(x) = 2$ and $M_q(y) = 3$. Consider two valuations $v : x = 1, y = 1$ and $v' : x = 0.5, y = 1.5$. We have $v \not\equiv_M v'$ according to Definition 5.2. But, notice that for any time elapse δ, whenever $v + \delta \models x \leq 2$, we have $v' + \delta \models x \leq 2$. This is because $v'(x) \leq v(x)$. The same property holds for the other guard $y \geq 3$: whenever $v + \delta \models y \geq 3$, we have $v' + \delta \models y \geq 3$. Now, this happens because $v'(y) \geq v(y)$. Therefore, it seems correct to say that v is simulated by v' when the guards are restricted to $\{x \leq 2, y \geq 3\}$. This gives a motivation to distinguish lower and upper bounds to get a coarser simulation relation.

7.2. LU-simulation

In an influential paper [Behrmann et al. 2004; 2006], the authors consider two bound functions L and U. The L-bound considers the maximum constant among relevant lower bound guards, and the U-bound considers the maximum constant among relevant upper bound guards. For example, for the set $\{x \leq 2, y \geq 3\}$ we will have $U(x) = 2, L(y) = 3$ and $L(x) = U(y) = -\infty$. A simulation relation $v \preccurlyeq_{LU} v'$ making use of the L and U bounds has been defined in [Behrmann et al. 2004; 2006]. Notice that for the same LU bounds as above, there could be several sets of guards. For example $\{x \leq 2, x \leq 1, y \geq 3, y \geq 0\}$ also has the same LU bounds. The simulation relation $v \preccurlyeq_{LU} v'$ caters to all possible guard sets that are compatible with given LU bounds. We will give some intuition for $\preccurlyeq_{LU}$ and then formally recall the definition of $v \preccurlyeq_{LU} v'$.

We will say that a constraint $x \sim c$ conforms to LU when $c \leq U(x)$ if $\sim \in \{<, \leq\}$ and $c \leq L(x)$ if $\sim \in \{>, \geq\}$. An automaton $\mathcal{A}$ conforms to LU-bounds if every constraint present in $\mathcal{A}$ conforms to LU. Suppose $v(x) > U(x)$. Then v can satisfy no upper constraints, even after time elapse. In this case we are fine with any value for $v'(x)$. Suppose $v(x) \leq U(x)$. Then, v could potentially satisfy some relevant upper bound constraint on x that conforms to U. It is safe to assume $v'(x) \leq v(x)$. This gives the condition: $v(x) \leq U(x)$ implies $v'(x) \leq v(x)$. Similarly, suppose $v(x) > L(x)$. Then v satisfies all the lower constraints. We need $v'(x) > L(x)$. Finally, if $v(x) \leq L(x)$, valuation v already satisfies some lower constraints currently, and so we assume $v'(x) \geq v(x)$. This culminates in the following definition.

Definition 7.1 (LU-*preorder*). Let $L : X \to \mathbb{N} \cup \{-\infty\}$ and $U : X \to \mathbb{N} \cup \{-\infty\}$ be two bounds functions. For valuations v and v', we define $v \preccurlyeq_{LU} v'$ if for all clocks x:

— $v(x) \leq U(x)$ implies $v'(x) \leq v(x)$
— $v(x) \leq L(x)$ implies $v'(x) \geq v(x)$
— $v(x) > L(x)$ implies $v'(x) > L(x)$

It can be shown that the LU-preorder induces a simulation relation for all automata that conform to the LU-bounds [Behrmann et al. 2006; Herbreteau et al. 2016]. This is why we call it the LU-simulation. Furthermore, it has been shown that the associated simulation test over zones $Z \preccurlyeq_{LU} Z'$ can be done in $\mathcal{O}(|X|^2)$ [Herbreteau et al. 2012; 2016]. Therefore one could very well use the LU-simulation to get a finite simulation graph, which at times is significantly smaller than the simulation graph using $\preccurlyeq_M$. It has also been proved that the LU-preorder is optimal while considering all automata conforming to given LU-bounds [Herbreteau et al. 2012; 2016]. Therefore, in order to do better, we need more information than just LU.

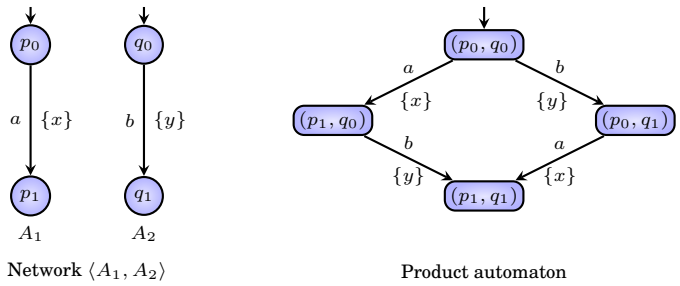


Fig. 10. A network of timed automata on the left, and its synchronized product on the right

When the LU-preorder was introduced [Behrmann et al. 2004; 2006], the efficient simulation test for $Z \preccurlyeq_{LU} Z'$ was not known. Instead the authors used an extrapolation operation on zones to get a finite graph. Extrapolations can work only in the presence of static constants whereas in the simulation graph approach, one can use dynamic constants as discussed in Section 6. In particular, the lazy bounds computation of [Herbreteau et al. 2013] uses LU constants.

8. A LOCAL-TIME SEMANTICS TO MAKE USE OF CONCURRENCY

In the entire discussion in the document so far, we have considered a monolithic timed automaton as the input. In practice, each component of the system is represented as a timed automaton. The system is then given as a network of timed automata which communicate with each other over shared actions. The reachability algorithm described until now considers a synchronized product of the network for the zone graph computation. All this is done on-the-fly, and so the whole product need not be computed upfront.

This framework does not exploit valuable information coming due to concurrency. For example, consider Figure 10. It shows a network and the corresponding synchronized product. Notice that both ab and ba lead to the same state $\langle p_1, q_1 \rangle$ in the product.

(1) A zone graph computation would lead to two different zones at the state $\langle p_1, q_1 \rangle$ due to the different order of clock resets. In general there could be exponentially many zones (in the number of components) at one control state of the network, plainly due to the different order of executing the same set of actions. Current methods can make use of simulations to cut out some of these zones, but that is only due to chance. The notion of simulations only looks at the future and is not meant to handle the issue of interleavings.
(2) In the untimed setting, the presence of diamonds due to independent actions has led to the study of partial-order reductions which have been an influential technique in explicit-state model checking. In the timed setting the clock resets break the diamonds, thereby creating a fundamental bottleneck for partial-order reductions.

To address these issues, a local-time semantics for networks of timed automata has been proposed [Bengtsson et al. 1998]. In the local-time semantics, each component in the network moves independently according to its local timeline which contrasts with the standard semantics where time elapses synchronously in all the components. When components perform a shared action, they synchronize their local timelines. This semantics gives good independence properties: for instance, if a and b are actions performed by components P_a and P_b, an execution $(a, 2)(b, 1)$ means a happens when the local time of P_a is 2 and b happens when local time of P_b is 1. There is no "happens-

before" between $(a, 2)$ and $(b, 1)$. The local-time semantics leads to a local-zone graph computation in which performing ab or ba from a local-zone leads to the same local-zone. This is very attractive since this settles (1) above and tempts us to consider (2), a partial-order reduction for timed automata. However, there is no free lunch. It turns out that it is impossible to get a finite simulation for the local-time semantics [Govind et al. 2022]. A way to prune the local-zone graph computation without using simulations was provided in [Govind et al. 2019]. This already leads to significant gains in practice due to the tackling of (1). The method proposed in [Govind et al. 2019] is not amenable to partial-order reduction. This problem is addressed in [Govind et al. 2022]. The impossibility of a finite simulation is an impediment for the application of partial-order methods. The current solution is to consider a restricted class of networks called bounded-spread networks [Govind et al. 2022]. For these networks, a simulation relation has been defined and a more sophisticated way to use it in the zone graph computation has been presented. The main advantage of this approach is that partial-order reductions (and finite simulations) can be applied. Investigating a suitable partial-order reduction method for the timed setting and evaluating the gains on the local-zone graph computation is an exciting line of work.

9. THE TOOL TCHECKER

The basic algorithm using zones and the $\preccurlyeq_{LU}$ simulation, the various ways to obtain constants and the algorithms using the local-time semantics have been implemented in the fully open-source tool TChecker [Herbreteau and Point 2019]. In addition to being a verification tool for real-time systems, TChecker has been a backbone for theoretical development, with several intuitions coming from experiments. The architecture of the tool allows easy extension to other verification algorithms and also to richer models of timed automata, like weighted timed automata, timed automata with diagonal constraints and updates, etc. A companion tool UPPAAL-To-TChecker [Point 2022b] can be used to translate a subset of the UPPAAL input language to the format of TChecker. The tool can also be used online in the TChecker demonstration page [Point 2022a].

10. CONCLUSION

We have presented an algorithmic framework for reachability in timed automata that uses zones and simulations (Sections 3 and 4). A concrete simulation relation $\preccurlyeq_M$ was presented in Section 5. We have presented $\preccurlyeq_M$ due to its simplicity. The simulation relation that is implemented in tools is $\preccurlyeq_{LU}$, which was presented in Section 7. In Section 6 we have described some methods to get smaller bounds, and hence better simulation parameters. Finally, in Section 8 we have touched upon an approach using a local-time semantics that makes use of concurrency.

The basic framework using zone graphs and simulations has been extended to richer models like timed automata with diagonal constraints and updates [Gastin et al. 2018; 2019; 2020], weighted timed automata [Bouyer et al. 2016], pushdown timed automata [Akshay et al. 2021], event-clock automata [Akshay et al. 2022], and for checking richer properties in timed automata like liveness [Herbreteau et al. 2020]. One promising direction is to investigate the simulation approach for other extensions like parametric timed automata [Alur et al. 1993; André 2021], timed games [Cassez et al. 2005; Behrmann et al. 2007], probabilistic timed automata [Norman et al. 2013; Kwiatkowska et al. 2011], etc., all of which have some current tool support.

Coming back to reachability in timed automata, where do we stand currently? Let us first get a sense of the scale. For a network consisting of 10 components, each with say 10 states and 1 clock, we already get a synchronized product with around 10^{10} control states and 10 clocks. The clocks may potentially lead to 2^{10} zones attached to each state. With current techniques, we have been able to handle a model with 140 components,

each with about 8 states and 3 clocks (see the Experiments section in [Herbreteau et al. 2013]). In all there are about 140^8 states and 420 clocks in this model, and the zone based algorithm (using lazy constants) is able to handle this scale. There is still a long way to go. Majority of the work has focussed on getting fewer zones per control state and this has seen quite some success. However, the size of the large control state space is a big burden. For untimed systems, symbolic methods using Binary Decision Diagrams (BDDs) and SAT have been used to tackle this large (discrete) state space. Enumerative algorithms have largely cashed in on partial-order reduction methods to optimize the control state space. Extending these methods to the timed setting does not seem straightforward.

For timed systems BDD and SAT based solutions [Møller et al. 1999; Behrmann et al. 1999; Wang 2001; Audemard et al. 2002; Niebert et al. 2002; Ehlers et al. 2010; Badban and Lange 2011; Roussanaly et al. 2019] have been tried with mixed results. Partial-order methods for timed automata have in general considered restricted settings: working with a subclass of networks [Govind et al. 2022], limiting partial-order methods only to parts where independent actions occur in zero-time [Møller et al. 1999; Larsen et al. 2020; Bønneland et al. 2021], or discovering which actions remain independent in the standard global-time semantics, either statically [Dams et al. 1998] or dynamically [Hansen et al. 2014]. We are not aware of any tool that has managed to successfully deploy partial-order reduction for timed automata. The recent study of the local-time semantics gives some hope in this direction.

REFERENCES

Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. 2006. Scheduling with timed automata. *Theor. Comput. Sci.* 354, 2 (2006), 272–300. DOI: http://dx.doi.org/10.1016/j.tcs.2005.11.018

S. Akshay, Paul Gastin, R. Govind, and B. Srivathsan. 2022. Simulations for Event-Clock Automata. In *Proceedings of CONCUR'22, the 33rd International Conference on Concurrency Theory (LIPIcs)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Warsaw, Poland, *To appear*.

S. Akshay, Paul Gastin, and Karthik R. Prakash. 2021. Fast Zone-Based Algorithms for Reachability in Pushdown Timed Automata. In *Proceedings of CAV'21, the 33rd International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Vol. 12759. Springer, Virtual event, 619–642. DOI: http://dx.doi.org/10.1007/978-3-030-81685-8_30

Rajeev Alur. 1991. *Techniques for automatic verification of real-time systems*. Ph.D. Dissertation. Stanford University, USA.

Rajeev Alur and David L. Dill. 1994. A Theory of Timed Automata. *Theor. Comput. Sci.* 126, 2 (1994), 183–235. DOI: http://dx.doi.org/10.1016/0304-3975(94)90010-8

Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. 1993. Parametric real-time reasoning. In *Proceedings of STOC'93, the 25th Annual ACM Symposium on Theory of Computing*. ACM, San Diego, California, USA, 592–601. DOI: http://dx.doi.org/10.1145/167088.167242

Rajeev Alur, Alon Itai, Robert P. Kurshan, and Mihalis Yannakakis. 1995. Timing Verification by Successive Approximation. *Inf. Comput.* 118, 1 (1995), 142–157. DOI: http://dx.doi.org/10.1006/inco.1995.1059

Étienne André. 2021. IMITATOR 3: Synthesis of Timing Parameters Beyond Decidability. In *Proceedings of CAV'21, the 33rd International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Vol. 12759. Springer, Virtual event, 552–565. DOI: http://dx.doi.org/10.1007/978-3-030-81685-8_26

Gilles Audemard, Alessandro Cimatti, Artur Kornilowicz, and Roberto Sebastiani. 2002. Bounded Model Checking for Timed Systems. In *Proceedings of FORTE'02, the IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (Lecture Notes in Computer Science)*, Vol. 2529. Springer, Texas, USA, 243–259. DOI: http://dx.doi.org/10.1007/3-540-36135-9_16

Bahareh Badban and Martin Lange. 2011. Exact Incremental Analysis of Timed Automata with an SMT-Solver. In *Proceedings of FORMATS'11, the 9th International Conference on Formal Modeling and Analysis of Timed Systems (Lecture Notes in Computer Science)*, Vol. 6919. Springer, Aalborg, Denmark, 177–192. DOI: http://dx.doi.org/10.1007/978-3-642-24310-3_13

Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim Guldstrand Larsen. 2003. Static Guard Analysis in Timed Automata Verification. In *Proceedings of TACAS'03, 9th International Conference on Tools*

and Algorithms for the Construction and Analysis of Systems (Lecture Notes in Computer Science), Vol. 2619. Springer, Warsaw, Poland, 254–277. DOI: http://dx.doi.org/10.1007/3-540-36577-X_18

Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. 2004. Lower and Upper Bounds in Zone Based Abstractions of Timed Automata. In *Proceedings of TACAS'04, the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (Lecture Notes in Computer Science)*, Vol. 2988. Springer, Barcelona, Spain, 312–326. DOI: http://dx.doi.org/10.1007/978-3-540-24730-2_25

Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. 2006. Lower and upper bounds in zone-based abstractions of timed automata. *Int. J. Softw. Tools Technol. Transf.* 8, 3 (2006), 204–215. DOI: http://dx.doi.org/10.1007/s10009-005-0190-0

Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. 2007. UPPAAL-Tiga: Time for Playing Games!. In *Proceedings of CAV'07, the 19th International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Vol. 4590. Springer, Berlin, Germany, 121–125. DOI: http://dx.doi.org/10.1007/978-3-540-73368-3_14

Gerd Behrmann, Kim Guldstrand Larsen, Justin Pearson, Carsten Weise, and Wang Yi. 1999. Efficient Timed Reachability Analysis Using Clock Difference Diagrams. In *Proceedings of CAV'99, the 11th International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Vol. 1633. Springer, Trento, Italy, 341–353. DOI: http://dx.doi.org/10.1007/3-540-48683-6_30

Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. 1998. Partial Order Reductions for Timed Systems. In *Proceedings of CONCUR'98, the 9th International Conference on Concurrency Theory (Lecture Notes in Computer Science)*, Vol. 1466. Springer, Nice, France, 485–500. DOI: http://dx.doi.org/10.1007/BFb0055643

Johan Bengtsson and Wang Yi. 2004. Timed automata: Semantics, algorithms and tools. In *Lecture Notes on Concurrency and Petri Nets*. Vol. 3098. Springer, Eichstätt, Germany, 87–124. DOI: http://dx.doi.org/10.1007/b98282

Patricia Bouyer. 2003. Untameable Timed Automata!. In *Proceedings of STACS'03, 20th Annual Symposium on Theoretical Aspects of Computer Science (Lecture Notes in Computer Science)*, Vol. 2607. Springer, Berlin, Germany, 620–631. DOI: http://dx.doi.org/10.1007/3-540-36494-3_54

Patricia Bouyer. 2004. Forward Analysis of Updatable Timed Automata. *Formal Methods Syst. Des.* 24, 3 (2004), 281–320. DOI: http://dx.doi.org/10.1023/B:FORM.0000026093.21513.31

Patricia Bouyer, Maximilien Colange, and Nicolas Markey. 2016. Symbolic Optimal Reachability in Weighted Timed Automata. In *Proceedings of CAV'16, the 28th International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Vol. 9779. Springer, Toronto, Canada, 513–530. DOI: http://dx.doi.org/10.1007/978-3-319-41528-4_28

Frederik Meyer Bønneland, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marco Muñiz, and Jiří Srba. 2021. Stubborn Set Reduction for Two-Player Reachability Games. *Logical Methods in Computer Science* Volume 17, Issue 1 (March 2021). DOI: http://dx.doi.org/10.23638/LMCS-17(1:21)2021

Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. 2005. Efficient On-the-Fly Algorithms for the Analysis of Timed Games. In *Proceedings of CONCUR'05, the 16th International Conference on Concurrency Theory (Lecture Notes in Computer Science)*, Vol. 3653. Springer, San Fransisco, California, USA, 66–80. DOI: http://dx.doi.org/10.1007/11539452_9

Costas Courcoubetis and Mihalis Yannakakis. 1992. Minimum and Maximum Delay Problems in Real-Time Systems. *Formal Methods Syst. Des.* 1, 4 (1992), 385–415. DOI: http://dx.doi.org/10.1007/BF00709157

Dennis Dams, Rob Gerth, Bart Knaack, and Ruurd Kuiper. 1998. Partial-order Reduction Techniques for Real-time Model Checking. *Formal Aspects Comput.* 10, 5-6 (1998), 469–482. DOI: http://dx.doi.org/10.1007/s001650050028

Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. 1995. The Tool KRONOS. In *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems (Lecture Notes in Computer Science)*, Vol. 1066. Springer, New Jersey, USA, 208–219. DOI: http://dx.doi.org/10.1007/BFb0020947

Conrado Daws and Stavros Tripakis. 1998. Model Checking of Real-Time Reachability Properties Using Abstractions. In *Proceedings of TACAS'98, the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems (Lecture Notes in Computer Science)*, Vol. 1384. Springer, Lisbon, Portugal, 313–329. DOI: http://dx.doi.org/10.1007/BFb0054180

David L. Dill. 1989. Timing Assumptions and Verification of Finite-State Concurrent Systems. In *Proceedings of International Workshop on Automatic Verification Methods for Finite State Systems*. Springer, Grenoble, France, 197–212. DOI: http://dx.doi.org/10.1007/3-540-52148-8_17

Rüdiger Ehlers, Daniel Fass, Michael Gerke, and Hans-Jörg Peter. 2010. Fully Symbolic Timed Model Checking Using Constraint Matrix Diagrams. In *Proceedings of RTSS'10, the 31st IEEE*

Real-Time Systems Symposium. IEEE Computer Society, San Diego, California, USA, 360–371. DOI:http://dx.doi.org/10.1109/RTSS.2010.36

John Fearnley and Marcin Jurdzinski. 2013. Reachability in two-clock timed Automata is PSPACE-Complete. In *Proceedings of ICALP'13, 40th International Conference on Automata, Languages, and Programming (Lecture Notes in Computer Science)*, Vol. 7966. Springer, Riga, Latvia, 212–223. DOI:http://dx.doi.org/10.1007/978-3-642-39212-2_21

John Fearnley and Marcin Jurdzinski. 2015. Reachability in two-clock timed automata is PSPACE-complete. *Inf. Comput.* 243 (2015), 26–36. DOI:http://dx.doi.org/10.1016/j.ic.2014.12.004

Paul Gastin, Sayan Mukherjee, and B. Srivathsan. 2018. Reachability in Timed Automata with Diagonal Constraints. In *Proceedings of CONCUR'18, the 29th International Conference on Concurrency Theory (LIPIcs)*, Vol. 118. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Beijing, China, 28:1–28:17. DOI:http://dx.doi.org/10.4230/LIPIcs.CONCUR.2018.28

Paul Gastin, Sayan Mukherjee, and B. Srivathsan. 2019. Fast Algorithms for Handling Diagonal Constraints in Timed Automata. In *Proceedings of CAV'19, the 31st International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Vol. 11561. Springer, New York City, USA, 41–59. DOI:http://dx.doi.org/10.1007/978-3-030-25540-4_3

Paul Gastin, Sayan Mukherjee, and B. Srivathsan. 2020. Reachability for Updatable Timed Automata Made Faster and More Effective. In *Proceedings of FSTTCS'20, the 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (LIPIcs)*, Vol. 182. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Goa, India (Virtual conference), 47:1–47:17. DOI:http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2020.47

R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. 2019. Revisiting Local Time Semantics for Networks of Timed Automata. In *Proceedings of CONCUR'19, the 30th International Conference on Concurrency Theory (LIPIcs)*, Vol. 140. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Amsterdam, the Netherlands, 16:1–16:15. DOI:http://dx.doi.org/10.4230/LIPIcs.CONCUR.2019.16

R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. 2022. Abstrations for the local-time semantics of timed automata: a foundation for partial-order methods. In *Proceedings of LICS'22, the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. Haifa, Israel. To appear.

Henri Hansen, Shang-Wei Lin, Yang Liu, Truong Khanh Nguyen, and Jun Sun. 2014. Diamonds Are a Girl's Best Friend: Partial Order Reduction for Timed Automata with Abstractions. In *Proceedings of CAV'14, the 26th International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Vol. 8559. Springer, Vienna, Austria, 391–406. DOI:http://dx.doi.org/10.1007/978-3-319-08867-9_26

Frédéric Herbreteau, Dileep Kini, B. Srivathsan, and Igor Walukiewicz. 2011. Using non-convex approximations for efficient analysis of timed automata. In *Proceedings of FSTTCS'11, IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (LIPIcs)*, Vol. 13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Mumbai, India, 78–89. DOI:http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2011.78

F. Herbreteau and G. Point. 2019. The TChecker tool and librairies. https://github.com/ticktac-project/tchecker. (2019).

Frédéric Herbreteau, B. Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz. 2020. Why Liveness for Timed Automata Is Hard, and What We Can Do About It. *ACM Trans. Comput. Log.* 21, 3 (2020), 17:1–17:28. DOI:http://dx.doi.org/10.1145/3372310

Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. 2012. Better Abstractions for Timed Automata. In *Proceedings LICS'12, the 27th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, Dubrovnik, Croatia, 375–384. DOI:http://dx.doi.org/10.1109/LICS.2012.48

Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. 2013. Lazy Abstractions for Timed Automata. In *Proceedings of CAV'13, the 25th International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Vol. 8044. Springer, Saint Petersburg, Russia, 990–1005. DOI:http://dx.doi.org/10.1007/978-3-642-39799-8_71

Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. 2016. Better abstractions for timed automata. *Inf. Comput.* 251 (2016), 67–90. DOI:http://dx.doi.org/10.1016/j.ic.2016.07.004

Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. 2015. LTSmin: High-Performance Language-Independent Model Checking. In *Proceedings of TACAS'15, the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (Lecture Notes in Computer Science)*, Vol. 9035. Springer, London, UK, 692–707. DOI:http://dx.doi.org/10.1007/978-3-662-46681-0_61

Sebastian Kupferschmid, Martin Wehrle, Bernhard Nebel, and Andreas Podelski. 2008. Faster Than Uppaal?. In *Proceedings of CAV'08, the 20th International Conference on Computer Aided Ver-*

ification (Lecture Notes in Computer Science), Vol. 5123. Springer, Princeton, USA, 552–555. DOI:http://dx.doi.org/10.1007/978-3-540-70545-1_53

Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proceedings of CAV'11, the 23rd International Conference on Computer Aided Verification (LNCS)*, Vol. 6806. Springer, Snowbird, UT, USA, 585–591. DOI:http://dx.doi.org/10.1007/978-3-642-22110-1

François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. 2004. Model Checking timed automata with one or two clocks. In *Proceedings of CONCUR'04, the 15th Conference on Concurrency Theory (Lecture Notes in Computer Science)*, Vol. 3170. Springer, London, UK, 387–401.

Kim G. Larsen, Marius Mikucionis, Marco Muñiz, and Jirí Srba. 2020. Urgent Partial Order Reduction for Extended Timed Automata. In *Proceedings of ATVA'20, the 18th International Symposium on Automated Technology for Verification and Analysis (Lecture Notes in Computer Science)*, Vol. 12302. Springer, Hanoi, Vietnam, 179–195. DOI:http://dx.doi.org/10.1007/978-3-030-59152-6_10

Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. 1997. UPPAAL in a Nutshell. *Int. J. Softw. Tools Technol. Transf.* 1, 1-2 (1997), 134–152. DOI:http://dx.doi.org/10.1007/s100090050010

Oded Maler and Amir Pnueli. 1995. Timing analysis of asynchronous circuits using timed automata. In *Proceedings of CHARME'95, Correct Hardware Design and Verification Methods, IFIP WG 10.5 Advanced Research Working Conference (Lecture Notes in Computer Science)*, Vol. 987. Springer, Frankfurt, Germany, 189–205. DOI:http://dx.doi.org/10.1007/3-540-60385-9

Jesper B. Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. 1999. Fully Symbolic Model Checking of Timed Systems using Difference Decision Diagrams. *Electron. Notes Theor. Comput. Sci.* 23, 2 (1999), 88–107. DOI:http://dx.doi.org/10.1016/S1571-0661(04)80671-6

Peter Niebert, Moez Mahfoudh, Eugene Asarin, Marius Bozga, Oded Maler, and Navendu Jain. 2002. Verification of Timed Automata via Satisfiability Checking. In *Proceedings of FTRTFT 2002, the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, Co-sponsored by IFIP WG 2.2 (Lecture Notes in Computer Science)*, Vol. 2469. Springer, Oldenburg, Germany, 225–244. DOI:http://dx.doi.org/10.1007/3-540-45739-9_15

Gethin Norman, David Parker, and Jeremy Sproston. 2013. Model checking for probabilistic timed automata. *Formal Methods Syst. Des.* 43, 2 (2013), 164–190. DOI:http://dx.doi.org/10.1007/s10703-012-0177-x

Gérald Point. 2022a. TChecker online demonstration. https://tchecker.labri.fr/. (2022).

Gérald Point. 2022b. UPPAAL-To-TChecker: a tool to translate UPPAAL models into TChecker models. https://github.com/ticktac-project/uppaal-to-tchecker. (2022).

Victor Roussanaly, Ocan Sankur, and Nicolas Markey. 2019. Abstraction Refinement Algorithms for Timed Automata. In *Proceedings of CAV'19, the 31st International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Vol. 11561. Springer, New York City, USA, 22–40. DOI:http://dx.doi.org/10.1007/978-3-030-25540-4_2

Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. 2009. PAT: Towards Flexible Verification under Fairness. In *Proceedings of CAV'09, the 21th International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Vol. 5643. Springer, Grenoble, France, 709–714. DOI:http://dx.doi.org/10.1007/978-3-642-02658-4_59

Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. 2017. Theta: a Framework for Abstraction Refinement-Based Model Checking. In *Proceedings FMCAD'17, the 17th Conference on Formal Methods in Computer-Aided Design*. IEEE, Vienna, Austria, 176–179. DOI:http://dx.doi.org/10.23919/FMCAD.2017.8102257

Stavros Tripakis and Sergio Yovine. 2001. Analysis of Timed Systems Using Time-Abstracting Bisimulations. *Formal Methods Syst. Des.* 18, 1 (2001), 25–68.

Farn Wang. 2001. Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram. In *Proceedings of FORTE'01, IFIP TC6/WG6.1 - 21st International Conference on Formal Techniques for Networked and Distributed Systems (IFIP Conference Proceedings)*, Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee (Eds.), Vol. 197. Kluwer, Cheju Island, Korea, 235–250.

Farn Wang. 2006. REDLIB for the Formal Verification of Embedded Systems. In *Leveraging Applications of Formal Methods, Second International Symposium, ISoLA 2006*. IEEE Computer Society, Paphos, Cyprus, 341–346. DOI:http://dx.doi.org/10.1109/ISoLA.2006.68

Jianhua Zhao, Xuandong Li, and Guoliang Zheng. 2005. A quadratic-time DBM-based successor algorithm for checking timed automata. *Information processing letters* 96, 3 (2005), 101–105.