Model Checking Timed Automata *

Sergio Yovine †‡
VERIMAG
Centre Equation
2, Av. de Vignate
38610 Gières, France

September 9, 1997

1 Introduction

The theory of *timed automata* provides a formal framework to model and to analyze the behavior of *real-time* systems, that is, of systems whose correct functioning is subject to and must ensure the respect of strict timing constraints such as execution times, response times, tasks' periods, communication delays and so on.

Timed automata have been first proposed in [4] as an extension of the automata-theoretic approach to the modeling of real-time systems. Since then, the theory of timed automata has been an intensive field of research in computer science. With the aim of demonstrating that the ultimate goal of the theoretical achievements is to apply them to solve real-life problems, the work on the theory of timed automata has been consistently accompanied by the development of tools.

1.1 Timed automata

A timed automaton is a finite-state machine equipped with a set of *clocks*. Clocks are piece-wise continuous real-valued functions of time that precisely record the time elapsed between events. All clocks are synchronized, that is, they all advance at the same pace. More precisely, all clocks have the same derivative with respect to time, which is assumed to be by definition equal to one. Discontinuities may occur when a transition is taken. In this case, clocks are allowed to be reset to a new value which becomes the initial value of the next

*To appear in LNCS volume on Embedded Systems, G. Rozenberg and F. Vaandrager, editors, Springer-Verlag, 1997.

†Sergio.Yovine@imag.fr. http://www.imag.fr/VERIMAG/PEOPLE/Sergio.Yovine

‡Currently visiting California PATH, University of California at Berkeley, Richmond Field Station Bldg. 452, 1301 S. 46th St, Richmond CA 94804.

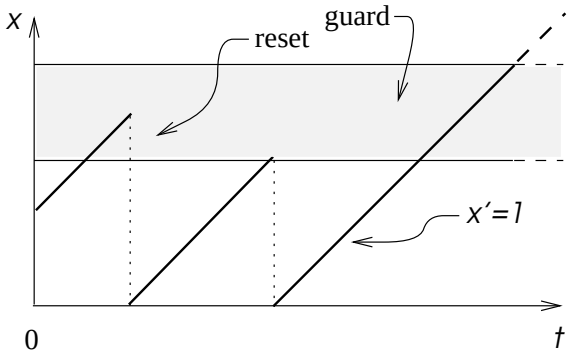


Figure 1: A clock (x) is a piece-wise continuous function of time (t).

continuous phase. Transitions are associated with a *guard* which is a predicate over the clocks. The guard determines when a transition can be taken. The behavior of a clock as a function of time is illustrated in Figure 1. Timed automata are formally presented in Section 2.

1.2 Algorithmic verification

The constraints imposed on the clocks model the timing constraints the real-time system is subject to. Analyzing the behavior of the timed automaton consists in verifying whether it satisfies the timing properties it is supposed to ensure. Such properties are generically called *requirements*. Assuming that we have a formal way of expressing the requirements, we can state the *verification problem* as follows. Given a timed automaton A, a requirement R and a *satisfaction relation* $\models$, does $A \models R$ hold?

Researchers focused their work on the development of algorithmic methods for solving this problem, mainly along the lines of the so-called *model checking* approach. This approach consists in providing an algorithm that answers yes or no for every instance of a class of verification questions. Such a class of problems is obtained by fixing the formalism to express the requirements and the satisfaction relation. Some examples of classes of verification problems are the following: requirements are also expressed as timed automata and the satisfaction relation corresponds to bisimulation equivalence [17, 34, 41, 43], or requirements are expressed as formulas of a logic and the satisfaction relation is "being a model of" [5, 3, 30, 12]. [1]

[1] As a matter of fact, it was this last class of problems that gave the name to the approach. Today, however, the term "model checking" is used as a synonym for "algorithmic verification", in contrast to "theorem proving" which is used to mean "deductive verification". [42]

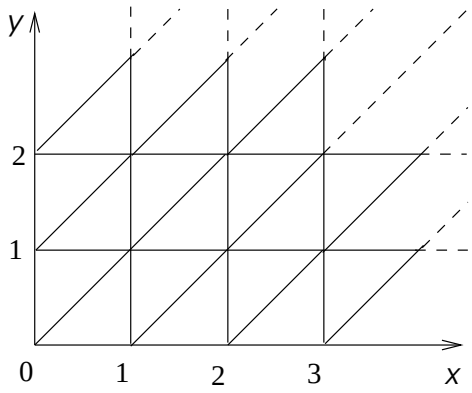


Figure 2: Partition of the space of clock valuations into regions.

1.3 Reachability

Many different instances of the verification question have been studied in the literature. In this paper we survey the different algorithms and data-structures that have been proposed in the seek for efficiency. For the sake of simplicity, we focus our attention on algorithms for solving the so-called *reachability problem*. This problem is stated as follows. Given two states of the system, is there an execution starting at one of them that reaches the other?

The first reason for studying such algorithms is that they allow us to check for *safety* properties, which are expressed as the non-reachability of a set of states where the system is consider to show an *incorrect* or *unsafe* functioning. Second, the algorithms developed for analyzing other classes of properties are essentially based on the same techniques and data-structures.

1.4 Algorithms

Verification algorithms can be viewed as algorithms that *search* for particular states among all the possible states of the system. States are related by a *transition relation* that specifies how to move from one state to another. That is, the state-space of a system is structured as a graph. Conceptually, the search can be performed in either of two ways: *forwards* or *backwards*. Forward search consists in traversing the state-space by moving from one state to its successors. The backward search analyzes the graph by exploring the predecessors of a state.

The state-space of a timed automaton is clearly infinite since clocks are variables that range over the set of real numbers. A very important observation in [4] is that the space of clock valuations, which is indeed dense, can be partitioned into a finite set of classes which are given the name of *clock regions*. The important property of the partition is that the verification question has the

same answer for all the clock valuations in a region. [2] This technique is studied in Section 3.

Algorithms directly based on the explicit construction of such a partition are however unlikely to perform efficiently in practice. The main reason for this is that the size of the partition, that is, the number of regions, is an exponential function of the number of clocks and of the constants against which the clocks are compared to. [3] Thus, different techniques have been proposed in order to overcome this problem.

Another approach consists in *discretizing* the space of clock valuations and the flow of time in such a way that at least one *representative* of every clock region belongs to the discretized space. Though this approach could suffer of the same exponential blow-up of the number of representatives, which is at least as large as the number of regions, its main advantage is the possibility of using well-known data-structures and algorithms that have been developed for the analysis of purely discrete systems. This approach is briefly discussed in Section 4.

An important corollary that follows from the results in [4] is that sets of regions can be characterized as conjunctions and disjunctions of linear inequalities over the clocks. Thus, a question that naturally arises is the following. Is it possible to analyze a timed automaton without explicitly and a-priori constructing the partition into regions, but rather by symbolically manipulating the inequalities? This problem has been explored in [30] in a more general verification problem than reachability, namely the verification of properties stated in the temporal logic called TCTL. Section 5 discusses the application of the techniques developed in [30] to solve the reachability problem. The verification question for TCTL is described in Appendix A.

The fourth and last approach over-viewed in this paper consists in finding a partition of the state-space that has two properties. First, it is *coarser*, and hence it has fewer classes, than the partition into regions. Second, for all the states in the same class, the verification question has the same answer. This approach is explained in Section 6.

1.5 Data structures and tools

A consequence of the results in [3, 30] is that convex sets of regions can be represented as conjunctions of inequalities involving only one clock or the difference

[2] In [4] this partition is constructed to solve the so-called *emptiness* problem. Given an initial state, is there at least one infinite execution starting from this state? Indeed, later developments showed that the partitioning of the space into regions is a fundamental property of timed automata: all the proofs of decidability and complexity of the verification problems considered in the literature rely on the construction of such a partition. One such proof for instance is the one provided in [3] concerning the question whether a timed automaton satisfies a specification given as a formula of the temporal logic TCTL.

[3] This complexity is an intrinsic problem of equipping automata with clocks, and it has to be added to the well-known problem of the combinatorial explosion regarding the analysis of automata.

between two of them. This observation lead to the study of a data-structure called Difference Bound Matrices (DBM) [22].[4]

The key idea behind the use of DBM's as a data-structure for encoding the space of clock valuations is to have a compact representation of sets of adjacent regions. However, a set containing regions that are apart from each other cannot be efficiently represented. Instead, a different approach consists in representing each region by one or more representative clock valuations. In this case, the partition is viewed as a finite set of points whose characteristic function can be encoded using Binary Decision Diagrams (BDD) [15]. Such an encoding leads to a compact representation of a sparse sets of regions.

These two data-structures and the tools that used them are discussed in Section 7.

2 Timed automata

2.1 Clocks

Let $\mathcal{X}$ be a finite set of variables called *clocks*. A *clock valuation* is a function that assigns a non-negative real-value to every clock. The set of valuations of $\mathcal{X}$, denoted $\mathcal{V}_{\mathcal{X}}$, is the set $[\mathcal{X} \stackrel{tot}{\to} \mathbb{R}^+]$ of total mappings from $\mathcal{X}$ to $\mathbb{R}^+$.

Let $v \in \mathcal{V}_{\mathcal{X}}$ and $\delta \in \mathbb{R}^+$. We denote by $v + \delta$ the clock valuation that maps each clock $x \in \mathcal{X}$ to the value $v(x) + \delta$.

Let $\mathcal{X}^*$ be the set $\mathcal{X} \cup \{0\}$. An *assignment* is a function that maps every clock into another clock or 0. The set of assignments over $\mathcal{X}$, denoted $?_{\mathcal{X}}$, is the set $[\mathcal{X} \stackrel{tot}{\to} \mathcal{X}^*]$.

Let $v \in \mathcal{V}_{\mathcal{X}}$ and $\gamma \in ?_{\mathcal{X}}$. We denote by $v[\gamma]$ the clock valuation such that for all $x \in \mathcal{X}$,

$$v[\gamma](x) = \begin{cases} v(\gamma(x)) & \text{if } \gamma(x) \in \mathcal{X}, \\ 0 & \text{otherwise.} \end{cases}$$

2.2 Clock constraints

The set $\Psi_{\mathcal{X}}$ of *clock constraints* over the set of clocks $\mathcal{X}$ is defined by the following grammar:

$$\psi \quad ::= \quad x \prec c \mid x - x' \prec c \mid \psi \wedge \psi \mid \neg\psi$$

where $x, x' \in \mathcal{X}$, $\prec \in \{<, \leq\}$ and $c \in \mathbb{Z}$.

For $\psi \in \Psi_{\mathcal{X}}$ we write $\mathsf{clk}(\psi)$ to denote the set of clocks that appear in ψ.

[4] As a matter of fact, DBM's were already used for the analysis of Timed Petri Nets in [37], though their use for analyzing timed automata required the development of new algorithms [45, 38, 1].

$\mathsf{c|k}(\psi)$ is inductively defined as follows:

$$\begin{array}{lcl} \mathsf{c|k}(x \prec c) & = & \{x\} \\ \mathsf{c|k}(x - x' \prec c) & = & \{x, x'\} \\ \mathsf{c|k}(\psi \wedge \psi') & = & \mathsf{c|k}(\psi) \cup \mathsf{c|k}(\psi') \\ \mathsf{c|k}(\neg\psi) & = & \mathsf{c|k}(\psi) \end{array}$$

Clock constraints are evaluated over clock valuations. A valuation $v \in \mathcal{V}_{\mathcal{X}}$ is said to *satisfy* the clock constraint $\psi \in \Psi_{\mathcal{X}}$, denoted $v \models \psi$, if

$$\begin{array}{lll} v \models x \prec c & \text{iff} & v(x) \prec c \\ v \models x - x' \prec c & \text{iff} & v(x) - v(x') \prec c \\ v \models \psi \wedge \psi' & \text{iff} & v \models \psi \quad \text{and} \quad v \models \psi' \\ v \models \neg\psi & \text{iff} & v \not\models \psi \end{array}$$

We denote by $[\![\psi]\!]$ the set of valuations that satisfy ψ, that is,

$$[\![\psi]\!] = \{v \in \mathcal{V}_{\mathcal{X}} \mid v \models \psi\}.$$

We denote by $\psi_s[x/x']$ the clock constraint obtained by replacing each occurrence of x by x' in ψ.

2.3 Timed automata

A *timed automaton* $\mathbf{A}$ is a tuple $\langle \mathcal{S}, \mathcal{X}, \Sigma, \mathcal{E}, I, \Pi, P \rangle$ where:

1. $\mathcal{S}$ is a finite set of *locations*. We distinguish a special location s_{init} which is refer to as the *initial* location of $\mathbf{A}$.
2. $\mathcal{X}$ is a finite set of *clocks*.
3. Σ is a finite set of *labels*.
4. $\mathcal{E}$ is a finite set of *edges*. Each edge e is a tuple $(s, \sigma, \psi, \gamma, s')$ where
   (a) $s \in \mathcal{S}$ is the *source*,
   (b) $s' \in \mathcal{S}$ is the *target*,
   (c) $\sigma \in \Sigma$ is the *label*,
   (d) $\psi \in \Psi_{\mathcal{X}}$ is the *enabling condition*, and
   (e) $\gamma \in ?\,\mathcal{X}$ is the *assignment*.
5. $I \in [\mathcal{S} \stackrel{tot}{\to} \Psi_{\mathcal{X}}]$. We refer to $I(s)$ as the *invariant* of s.
6. Π is a finite set of *atomic propositions*.
7. $P \in [\mathcal{S} \stackrel{tot}{\to} \mathbf{2}^{\Pi}]$. We refer to $P(s)$ as the set of atomic propositions of s.

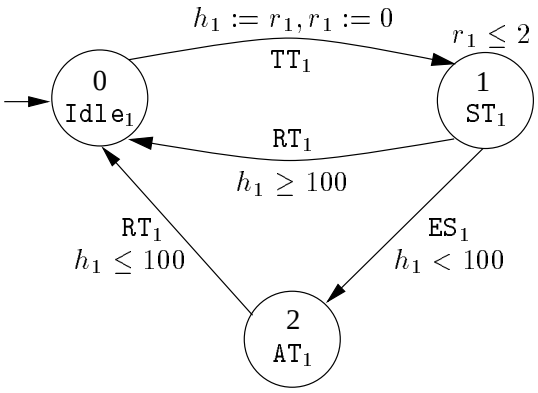


Figure 3: Timed automaton of a FDDI's sender station.

Example 1. Figure 3 shows a timed automaton with three locations labeled $\mathtt{Idle}_1$, $\mathtt{ST}_1$ and $\mathtt{AT}_1$, and two clocks, namely h_1 and r_1. This timed automaton models the behavior of a FDDI's sender station [31]. $\mathtt{Idle}_1$ is the initial location of the timed automaton. By definition, the initial value of a clock is 0. In location $\mathtt{Idle}_1$, the station is waiting for the token. The arrival of the token is model by the transition labeled $\mathtt{TT}_1$. Clock r_1 counts the time elapsed since the last reception of the token. Each time the token is received, r_1 is reset to 0 and h_1 is assigned the value of r_1. All the assignments are executed in parallel, that is, the order in which the list of assignments is given is meaningless. In location $\mathtt{ST}_1$, the station is sending high-priority messages. This phase can last at most 2 time units. This timing constraint is modeled by the invariant condition $r_1 \leq 2$. The station terminates the transmission either because 2 time units have elapsed or because it has no more high-priority messages to send. If more than 100 time units have elapsed since the previous reception of the token, the station goes back to location $\mathtt{Idle}_1$, otherwise, it goes to location $\mathtt{AT}_1$ where it can send low-priority messages while the value of the clock h_1 is less than 100. When the station has no more low-priority messages to send, it releases the token. □

2.4 States and transitions

The meaning of $\mathbf{A}$ is an infinite *transition system* $\langle \mathcal{Q}, \rightarrow, P^* \rangle$, where $\mathcal{Q}$ is the set of *states*, $\rightarrow$ is the *transition relation*, and P^* is the extension of P to states. A *state* of $\mathbf{A}$ is given by a location and a valuation of the clocks. At any state, $\mathbf{A}$ can move along one of the outgoing edges or it can remain in the location while time passes. In the first case, the transition results in a new state whose location is the target location of the edge and the valuation is modified according to the assignment. Such a transition is called a *discrete* transition, and may only happen when the valuation satisfies the enabling condition of the edge. Idling in a location during some time results in the values of the

clocks to be updated by the amount of time elapsed. Such transitions are called *timed* transitions. While remaining in a given location, $\mathbf{A}$ must respect the corresponding invariant condition.

Formally, $\langle \mathcal{Q}, \rightarrow, P^* \rangle$ is defined as follows:

1. $\mathcal{Q} = \{(s, v) \in \mathcal{S} \times \mathcal{V}_{\mathcal{X}} \mid v \models I(s)\}$. The initial state q_{init} is the pair (s_{init}, v_{init}) where $v_{init}(x) = 0$ for all $x \in \mathcal{X}$. We denote by $q + \delta$ the state $(s, v + \delta)$, and by $q[\gamma]$ the state $(s, v[\gamma])$.

2. $P^*(s, v) = P(s)$.

3. The transition relation $\rightarrow \subseteq \mathcal{Q} \times (\Sigma \cup \mathbb{R}^+) \times \mathcal{Q}$ is defined by the following rules:

$$\frac{e = (s, \sigma, \psi, \gamma, s') \in \mathcal{E} \quad v \models \psi \quad v[\gamma] \models I(s')}{(s, v) \xrightarrow{\sigma} (s', v[\gamma])} \tag{1}$$

$$\frac{\delta \in \mathbb{R}^+ \quad \forall \delta' \in \mathbb{R}^+.\ \delta' \leq \delta \Rightarrow (s, v + \delta') \models I(s)}{(s, v) \xrightarrow{\delta} (s, v + \delta)} \tag{2}$$

2.5 Executions

An *execution* or *run* r of $\mathbf{A}$ is an infinite sequence of states and transitions:

$$r \quad = \quad q_0 \xrightarrow{\ell_0} q_1 \xrightarrow{\ell_1} \cdots$$

We denote by $\mathcal{R}$ the set of runs of $\mathbf{A}$ and by $\mathcal{R}(q)$ the set of runs starting at the state $q \in \mathcal{Q}$.

A *position* p of r is a pair $(i, \delta) \in \mathbb{N} \times \mathbb{R}^+$ such that $\delta = 0$ if $\ell_i \in \Sigma$, otherwise $\delta = \ell_i$. We denote by $\mathcal{P}_r$ the set of positions of r. For a given $i \geq 0$, the set of positions of the form (i, δ) characterizes the set of states through which the run r passes while time flows from state q_i to state q_{i+1}. We define $\Xi(s_i, v_i)$ to be the state $(s_i, v_i + \delta)$. We define a total order $\ll$ on $\mathcal{P}_r$ as follows:

$$(i, \delta) \ll (j, \delta') \quad \text{iff} \quad i < j \vee (i = j \wedge \delta \leq \delta').$$

We define $\Delta(i)$ to be the time elapsed from state q_0 to state q_i. $\Delta(i)$ is inductively defined as follows:

$$\begin{aligned} \Delta(0) &= 0 \\ \Delta(i+1) &= \begin{cases} 0 & \text{if } \ell_i \in \Sigma, \\ \ell_i & \text{otherwise.} \end{cases} \end{aligned}$$

We define $\Delta(i, \delta)$ to be $\Delta(i) + \delta$.

A run r is said to be *time-divergent* if $\lim_{i \to \infty} \Delta(i) = \infty$. We denote by $\mathcal{R}^\infty$ the set of time-divergent runs of $\mathbf{A}$ and by $\mathcal{R}^\infty(q)$ the set of runs starting at the state $q \in \mathcal{Q}$.

A state q' is *reachable* from state q if it belongs to some run starting at q. We define $\mathsf{Reach}(q)$ to be the set of states reachable from q. That is,

$$\mathsf{Reach}(q) \quad = \quad \{\Xi(p) \mid \exists r \in \mathcal{R}(q).\ p \in \mathcal{P}_r\}.$$

We define $\mathsf{Reach}^{\infty}(q)$ to be the set of states reachable from q along some time-divergent run. That is,

$$\mathsf{Reach}^{\infty}(q) \quad = \quad \{\Xi(p) \mid \exists r \in \mathcal{R}^{\infty}(q).\ p \in \mathcal{P}_r\}.$$

A is said to be *Non-Zeno* or *well-timed* if for all states q, $\mathsf{Reach}^{\infty}(q) \neq \emptyset$. That is, every state can let time progress without bound.

Property 1. If **A** is *Non-Zeno*, $q' \in \mathsf{Reach}^{\infty}(q)$ iff $q' \in \mathsf{Reach}(q)$.

3 Analysis using the region graph

Given an initial state, we are interested in computing the set of states that are *reachable* from that state, that is, all the states that belong to some execution r starting at the initial state. In this section we show how to do so by *partitioning* the space of clock valuations.

3.1 Region equivalence

Let $\hat{\Psi} \subseteq \Psi_{\mathcal{X}}$ be a set of clock constraints over $\mathcal{X}$. For all $x \in \mathcal{X}$, let C_x be the biggest constant $c \in \mathbb{N}$ such that either $x \prec c$ or $x - y \prec c$ is a sub-formula of a clock constraint in $\hat{\Psi}$. We define $\simeq_{\hat{\Psi}} \subseteq \mathcal{V}_{\mathcal{X}} \times \mathcal{V}_{\mathcal{X}}$ to be the largest *reflexive* and *symmetric* relation such that $v \simeq_{\hat{\Psi}} v'$ iff for all $x, y \in \mathcal{X}$, the following three conditions hold:

1. $v(x) > C_x$ implies $v'(x) > C_x$,
2. if $v(x) \leq C_x$ then
   (a) $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, and
   (b) $\vartheta(v(x)) = 0$ implies $\vartheta(v'(x)) = 0$,

   where $\lfloor \cdot \rfloor : \mathbb{R}^+ \to \mathbb{N}$ and $\vartheta(\cdot) : \mathbb{R}^+ \to [0, 1)$, such that for $\delta \in \mathbb{R}^+$, $\lfloor \delta \rfloor$ is the integer part of δ, and $\vartheta(\delta)$ its fractional part.
3. for all clock constraints of the form $x - y \prec c$ with $c \in \mathbb{N}$ and $c \leq C_x$, $v \models x - y \prec c$ implies $v' \models x - y \prec c$.

Property 2. It is not difficult to prove that $\simeq_{\hat{\Psi}}$ is an *equivalence* relation with a finite number of classes.

$\simeq_{\hat{\Psi}}$ is called the *region* equivalence for the set of clock constraints $\hat{\Psi}$. We denote by $[v]$ the equivalence class (or region) of v.

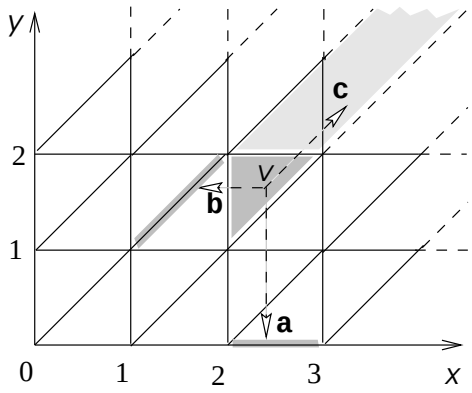


Figure 4: Properties of the region equivalence.

Example 2. Figure 2 illustrates the region equivalence for two clocks x and y with $C_x = 3$ and $C_y = 2$. □

The region equivalence has the following properties.

Property 3. Every region can be characterized by a clock constraint. □

Property 4. The number of regions is of the order of $n!2^n \Pi_{x \in \mathcal{X}} C_x$, where n is the number of clocks. □

Property 5. Let $v \simeq_{\hat{\Psi}} v'$. For every $\psi \in \hat{\Psi}$, $v \models \psi$ iff $v' \models \psi$. □

Property 6. Let $\hat{\Psi}$ be the set of all clock constraints appearing in $\mathbf{A}$, and let $v \simeq_{\hat{\Psi}} v'$.

1. For all $\gamma \in ?_{\mathcal{X}}$, $v[\gamma] \simeq_{\hat{\Psi}} v'[\gamma]$.
2. For all $\delta \in \mathbb{R}^+$, there exists $\delta' \in \mathbb{R}^+$, such that $v + \delta \simeq_{\hat{\Psi}} v' + \delta'$.

□

Property 7. Let ρ be a region such that for all v such that $[v] = \rho$, $v(x) > C_x$ for all $x \in \mathcal{X}$. Clearly, $[v + \delta] = \rho$ for all $\delta \in \mathbb{R}^+$. Such a region is said to be *unbounded* because the values of the clocks may grow without bound. □

Example 3. Figure 4 illustrates those properties. Consider the region defined by the clock constraint $2 < x < 3 \wedge 1 < y < 2 \wedge x - y < 1$. Let v be any clock valuation in this region.

a. Consider the assignment $y := 0$. The clock valuation $v[y := 0]$ belongs to the region $2 < x < 3 \wedge y = 0$.

b. Consider the assignment $x := y$. The clock valuation $v[x := y]$ belongs to the region $1 < x < 2 \wedge 1 < y < 2 \wedge x = y$.

c. Each time successor of v belongs to some of the regions crossed by a straight line drawn in the direction of the arrow.

□

3.2 Region graph

Let $(\mathcal{Q}, \rightarrow)$ be the transition system of $\mathbf{A}$. We extend the region equivalence $\simeq_{\hat{\Psi}}$ to the states of $\mathcal{Q}$ as follows. Two states $q = (s, v)$ and $q' = (s', v')$ are *region-equivalent*, denoted $q \simeq_{\hat{\Psi}} q'$, iff $s = s'$ and $v \simeq_{\hat{\Psi}} v'$. We denote by $[q]$ the equivalence class of q.

The region equivalence over states has the following properties.

Property 8. Let $\hat{\Psi}_{\mathbf{A}}$ be the set of all clock constraints appearing in $\mathbf{A}$, and let $q_1 \simeq_{\hat{\Psi}} q_2$.

1. For all $\sigma \in \Sigma$, whenever $q_1 \xrightarrow{\sigma} q_1'$ for some q_1', there exists q_2' such that $q_2 \xrightarrow{\sigma} q_2'$ and $q_2 \simeq_{\hat{\Psi}} q_2'$.

2. For all $\delta \in \mathbb{R}^+$, whenever $q_1 \xrightarrow{\delta} q_1'$ for some q_1', there exists q_2' and $\delta' \in \mathbb{R}^+$ such that $q_2 \xrightarrow{\delta'} q_2'$ and $q_2 \simeq_{\hat{\Psi}} q_2'$.

□

Therefore, if some state q_1' is reachable from q_1, a region-equivalent state q_2' is reachable from q_2.

Let $\hat{\Psi} \subseteq \Psi_{\mathcal{X}}$ be a set of clock constraints, $\hat{\Psi}_{\mathbf{A}}$ be the set of clock constraints of $\mathbf{A}$, and $\simeq$ be the region equivalence defined over $\hat{\Psi} \cup \hat{\Psi}_{\mathbf{A}}$.

The *region graph* $\mathsf{RG}(\mathbf{A}, \hat{\Psi})$ is the transition system $(\mathcal{Q}_{\simeq}, \rightarrow)$, where:

1. $\mathcal{Q}_{\simeq} = \{[q] \mid q \in \mathcal{Q}\}$.

2. $\rho \xrightarrow{\sigma} \rho'$ iff there exists $q, q' \in \mathcal{Q}$ such that $\rho = [q]$, $\rho' = [q']$, and $q \xrightarrow{\sigma} q'$.

3. $\rho \xrightarrow{\varepsilon} \rho'$ iff

   (a) $\rho = \rho'$ is an unbounded region, or

   (b) $\rho \neq \rho'$ and there exists $q \in \mathcal{Q}$ and $\delta \in \mathbb{R}^+$ such that $q \xrightarrow{\delta} q'$, and $\rho = [q]$, $\rho' = [q + \delta]$, and $q \xrightarrow{\delta} q'$, and for all $\delta' \in \mathbb{R}^+$, if $\delta' \leq \delta$ then $[q + \delta']$ is either ρ or ρ'.

Notice that only unbounded regions have self-loops labeled by ε. Thus, these loops represent the divergence of time at a location.

We define $\mathsf{Reach}(\rho)$ to be the set of regions reachable from the region ρ:

$$\mathsf{Reach}(\rho) \quad = \quad \{\rho' \mid \rho \rightarrow^* \rho'\}$$

where $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$.

We denote by $\langle q \rangle$ any clock constraint $\psi \in \Psi$ such that $q \models \psi$, and for all $\psi' \in \Psi$, if $q \models \psi'$ then ψ implies ψ'. That is, $\langle q \rangle$ is the *tightest* (modulo equivalence) clock constraint that characterizes the values of the clocks in q. Now, the question whether the state q' is reachable from the state q can be answered as follows.

Property 9. Let $\mathbf{A}$ be a timed automaton, $q, q' \in \mathcal{Q}$, and let $\mathsf{RG}(\mathbf{A}, \{\langle q \rangle, \langle q' \rangle\})$ be the corresponding region graph.

$$q' \in \mathsf{Reach}(q) \quad \text{iff} \quad [q'] \in \mathsf{Reach}([q]).$$

Notice that the constraints $\langle q \rangle$ and $\langle q' \rangle$ characterize exactly the equivalence classes $[q]$ and $[q']$, respectively. □

Property 9 says that the reachability problem for q and q' has the same answer for all the states which are region-equivalent to them.

Property 10. Because of Property 1, if $\mathbf{A}$ is *Non-Zeno*, Property 9 also holds for reachability along time-divergent executions.

3.3 Region-graph based algorithms

The last property says that verifying whether q' is reachable from q is decidable. Indeed, it is possible to find the answer to such question by traversing the region graph. There are basically two ways of doing so.

Forward traversal. This method consists in starting from $[q]$ and visiting the set of its successors and the successors of those and so on, until all the reachable regions have been visited. In other words, it consists in constructing the sequence of sets of regions $F_0 \subseteq F_1 \subseteq \cdots$, such that

$$\begin{aligned} F_0 &= [q] \\ F_{i+1} &= F_i \cup \mathsf{Suc}(F_i) \end{aligned}$$

where $\mathsf{Suc}(F_i) = \{\rho \mid \exists \rho_i \in F_i.\ \rho_i \rightarrow \rho\}$.

Property 11. $[q'] \in \mathsf{Reach}([q])$ iff $[q'] \in \bigcup_{i \geq 0} F_i$. □

Backward traversal. This method consists in starting from $[q']$ and visiting the set of its predecessors and the predecessors of those and so on, until all the regions from which is possible to reach $[q']$ have been visited. It consists then in constructing the sequence of sets of regions $B_0 \subseteq B_1 \subseteq \cdots$, such that

$$\begin{aligned} B_0 &= [q'] \\ B_{i+1} &= B_i \cup \mathsf{Pre}(B_i) \end{aligned}$$

where $\mathsf{Pre}(B_i) = \{\rho \mid \exists \rho_i \in B_i.\ \rho \rightarrow \rho_i\}$.

Property 12. $[q'] \in \mathsf{Reach}([q])$ iff $[q] \in \bigcup_{i \geq 0} B_i$. □

4 Analysis using representatives

Let $\mathcal{X} = \{x_1, \cdots, x_n\}$. Each region of the region graph can be represented by associating with every clock $x \in \mathcal{X}$, either a constant $c \in \{0, \cdots, C_x\}$, an interval $(c-1, c)$ with $c \in \{1, \cdots, C_x\}$, or the interval (C_x, ∞), and an ordering $x_{i_1} \#_{i_1} \cdots \#_{i_n} x_{i_n}$, with $\# \in \{<, \leq, =\}$, that encodes the ordering of the fractional parts.

Another approach consists in representing a region by one or more representatives. This method is equivalent to *discretizing* the space of clock valuations and the flow of time in such a way that at least one representative of every clock region belongs to the discretized space.

4.1 Properties

Discretizing the state space consists in taking a rational constant d, cutting the space into a d-grid, and providing a transition relation between the points of the grid. The discretization and the transition relation must satisfy the following conditions.

1. If a region ρ has a transition to a region ρ', then all the points of the grid that belong to ρ have a transition to some point that belongs to ρ'.

2. If a point of the grid has a transition to another point, then the region to which the former belongs has a transition to the region in which the latter resides.

These conditions ensure that the transition relation over the discretized state-space has the same properties than the region graph.

We denote by $\mathsf{rep}(\rho)$ the set of representatives of region ρ, and by $(\mathcal{D}, \rightarrow)$ the discretized graph, and by d the elements of $\mathcal{D}$. We define $\mathsf{Reach}(d)$ to be the set $\{d' \mid d \rightarrow^* d'\}$.

Property 13. $\rho' \in \mathsf{Reach}(\rho)$ iff $\mathsf{rep}(\rho') \cap \mathsf{Reach}(\mathsf{rep}(\rho)) \neq \emptyset$.

4.2 Algorithms

The algorithms for computing $\mathsf{Reach}(d)$ are basically the same used before to compute $\mathsf{Reach}(\rho)$.

Forward traversal. This method consists in computing the sequence of sets of representatives $F_0 \subseteq F_1 \subseteq \cdots$, such that

$$\begin{aligned} F_0 &= \mathsf{rep}([q]) \\ F_{i+1} &= F_i \cup \mathsf{Suc}(F_i) \end{aligned}$$

where $\mathsf{Suc}(F_i) = \{d \mid \exists d_i \in F_i.\ d_i \rightarrow d\}$.

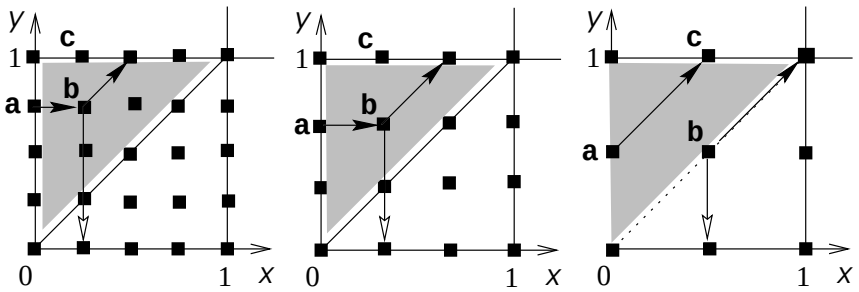


Figure 5: Discretizations of the space of clock valuations.

Backward traversal. This method consists in computing the sequence of sets of representatives $B_0 \subseteq B_1 \subseteq \cdots$, such that

$$\begin{aligned} B_0 &= \mathsf{rep}([q']) \\ B_{i+1} &= B_i \cup \mathsf{Pre}(B_i) \end{aligned}$$

where $\mathsf{Pre}(B_i) = \{d \mid \exists d_i \in B_i.\ d \to d_i\}$.

4.3 Discretization schemes

In [26] two discretization schemes have been proposed with $\epsilon = 1/(n+1)$ and $\epsilon = (1/2n)$. Figures 5(a) and 5(b) illustrate these two discretizations. Notice that for $(\mathcal{D}, \to)$ to satisfy the properties stated before, some "adjustments" need to be done to the computation of the time-successors. For instance, the time-successor of point `a` in Figures 5(a) and 5(b), is not point `c` as we could imagine, but point `b`. This is because when moving from `a` to `c` we *miss* the region represented by `c`.

In [8, 14] a simpler discretization has been proposed based on the following observation. The special class of timed automata where all the clock conditions are of the form $x \geq c$ or $x < c$, admits a slightly simpler and coarser region graph [29]. For these automata, a discretization with $\epsilon = 1/n$, where the passage of time is simply the addition of ϵ to all the clocks, is sufficient (see Figure 5(c)).

5 Analysis using clock constraints

Let F be the set of regions $\bigcup_{i \geq 0} F_i$ computed by the forward traversal algorithm explained in Section 3. Then F can be characterized as a disjoint union of the form $\biguplus_{s \in \mathcal{S}} F_s$, where F_s is the clock constraint that characterizes the set of regions that belong to F whose location is equal to s. The same observation holds for B. Indeed, such characterization can be computed without a-priori constructing the region graph.

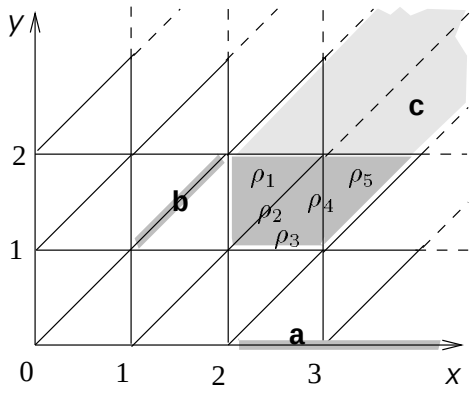


Figure 6: Representation of sets of regions as clock constraints.

Example 4. Consider the example illustrated in Figure 6. Let ψ be the clock constraint $1 < y < 2 \wedge 2 < x \wedge x - y < 2$. ψ represents the union of 5 regions.

`a.` Consider the assignment $y := 0$. The set of regions obtained by setting to 0 the value of y in all the clock valuations $v \in [\![\psi]\!]$ is characterized by the clock constraint $2 < x \wedge y = 0$.

`b.` Consider the assignment $x := y$. The set of regions obtained by assigning to x the value of y in all the clock valuations $v \in [\![\psi]\!]$ is characterized by the clock constraint $1 < x < 2 \wedge 1 < y < 2 \wedge x = y$.

`c.` The set of regions corresponding to the clock valuations reachable from the clock valuations in $[\![\psi]\!]$ by letting time pass is characterized by the clock constraint $1 < y \wedge x - y < 2 \wedge y - x < 0$.

This example suggests that we can actually compute the set of successor regions by symbolically manipulating the clock constraints. The same argument holds for predecessor regions. □

5.1 Forward computation of clock constraints

Let $s \in \mathcal{S}$, $\psi_s \in \Psi_{\mathcal{X}}$ and $e = (s, \psi, \sigma, \gamma, s') \in \mathcal{E}$. We denote by $\mathsf{Suc}_e(\psi_s)$ the predicate over $\mathcal{X}$ that characterizes the set of clock valuations that are reachable from the clock valuations in ψ_s when the timed automaton executes the discrete transition corresponding to the edge e. That is,

$$v \models \mathsf{Suc}_e(\psi_s) \quad \text{iff} \quad \exists v' \in \mathcal{Q}.\ v = v'[\gamma] \wedge v' \models (\psi_s \wedge \psi).$$

Property 14. $\mathsf{Suc}_e(\psi_s) \in \Psi_{\mathcal{X}}$. □

Example 5. Consider again the example illustrated in Figure 6. Recall that ψ is the clock constraint $1 < y < 2 \wedge 2 < x \wedge x - y < 2$.

a. The result of executing the transition resetting x to 0 is computed as follows.

$$\begin{aligned}
&\mathsf{Suc}_{\mathtt{a}}(\psi_s) = \\
&\quad = \quad \exists x', y' .\ \psi_s[x/x', y/y'] \wedge y = 0 \wedge x = x' \\
&\quad = \quad \exists x', y' .\ 1 < y' < 2 \wedge 2 < x' \wedge x' - y' < 2 \wedge y = 0 \wedge x = x' \\
&\quad = \quad \exists y' .\ 1 < y' < 2 \wedge 2 < x \wedge x - y' < 2 \wedge y = 0 \\
&\quad = \quad 2 < x \wedge x < 4 \wedge y = 0
\end{aligned}$$

Since the upper bound of 4 is greater than the constant $C_x = 3$, we can eliminate the clock constraint $x < 4$ and obtain: $\mathsf{Suc}_{\mathtt{a}}(\psi_s) = 2 < x \wedge y = 0$.

b. Now, consider the assignment $x := y$.

$$\begin{aligned}
&\mathsf{Suc}_{\mathtt{b}}(\psi_s) = \\
&\quad = \quad \exists x', y' .\ \psi_s[x/x', y/y'] \wedge y = y' \wedge x = y' \\
&\quad = \quad \exists x', y' .\ 1 < y' < 2 \wedge 2 < x' \wedge x' - y' < 2 \wedge y = y' \wedge x = y' \\
&\quad = \quad \exists x' .\ 1 < y < 2 \wedge 2 < x' \wedge x' - y < 2 \wedge x = y \\
&\quad = \quad 1 < y < 2 \wedge 0 < y \wedge x = y \\
&\quad = \quad 1 < y < 2 \wedge x = y
\end{aligned}$$

□

In other words, to compute $\mathsf{Suc}_e(\psi_s)$ is equivalent to visit all the regions that are e-successors of the regions in ψ_s, but without having to explicitly represent each one of them.

Let $s \in \mathcal{S}$ and $\psi_s \in \Psi_{\mathcal{X}}$. We denote by $\mathsf{Suc}_\varepsilon(\psi_s)$ the predicate over $\mathcal{X}$ that characterizes the set of clock valuations that are reachable from the clock valuations in ψ_s when the timed automaton lets time pass at s. That is,

$$v \models \mathsf{Suc}_\varepsilon(\psi_s) \quad \text{iff} \quad \exists \delta \in \mathbb{R}^+ .\ v - \delta \models \psi_s \wedge \forall \delta' \in \mathbb{R}^+ .\ \delta' \leq \delta \Rightarrow v - \delta' \models I(s).$$

Property 15. $\mathsf{Suc}_\varepsilon(\psi_s) \in \Psi_{\mathcal{X}}$. □

Example 6. Consider again the example illustrated in Figure 6. Case c corresponds to letting time pass at the location. For simplicity, we assume here that the invariant condition is true.

$$\begin{aligned}
&\mathsf{Suc}_\varepsilon(\psi_s) = \\
&\quad = \quad \exists \delta \in \mathbb{R}^+ .\ \psi_s[x/x - \delta, y/y - \delta] \\
&\quad = \quad \exists \delta \in \mathbb{R}^+ .\ 1 < y - \delta < 2 \wedge 2 < x - \delta \wedge (x - \delta) - (y - \delta) < 2 \\
&\quad = \quad \exists \delta \in \mathbb{R}^+ .\ 1 < y - \delta < 2 \wedge 2 < x - \delta \wedge x - y < 2 \wedge \\
&\quad = \quad 1 < y \wedge 2 < x \wedge y - x < 0 \wedge x - y < 2
\end{aligned}$$

□

Notice that $\mathsf{Suc}_\varepsilon(\psi_s)$ characterizes the set of the regions that contains the regions characterized by ψ_s and the regions reachable from them by taking only ε-transitions.

Now, we can solve the reachability problem by computing the sequence of sets of clock constraints $F_0, F_1, \cdots$ as follows:

$$\begin{aligned} F_0 &= \langle q \rangle \\ F_{i+1} &= \biguplus_{s \in \mathcal{S}} \left(\mathsf{Suc}_\varepsilon(F_{i,s}) \ \uplus \ \biguplus_{e \in \mathcal{E}} \mathsf{Suc}_e(F_{i,s}) \right) \end{aligned}$$

Notice that $F_{i,s}$ implies $F_{i+1,s}$ for all $i \geq 0$ and $s \in \mathcal{S}$.

Property 16. Let $F = \bigcup_{i \geq 0} F_i$, $q = (s, v)$, and $q' = (s', v')$. $[q'] \in \mathsf{Reach}([q])$ iff $\langle q' \rangle$ implies $F_{s'}$. □

5.2 Backward computation of clock constraints

Let $s \in \mathcal{S}$, $\psi_s \in \Psi_{\mathcal{X}}$ and $e = (s', \psi, \sigma, \gamma, s) \in \mathcal{E}$. We denote by $\mathsf{Pre}_e(\psi_s)$ the predicate over $\mathcal{X}$ that characterizes the set of clock valuations that can reach a clock valuation in ψ_s when the timed automaton executes the discrete transition corresponding to the edge e. That is,

$$v \models \mathsf{Pre}_e(\psi_s) \quad \text{iff} \quad v \models \psi \wedge v[\gamma] \models \psi_s$$

Property 17. $\mathsf{Pre}_e(\psi_s) \in \Psi_{\mathcal{X}}$. □

Example 7. Consider the example illustrated in Figure 7. Let ψ_s be the clock constraint $0 \leq y < 2 \wedge 1 < x < 2$.

a. Suppose that the edge we are considering has a guard ψ given by the constraint $y > x + 1$. The predecessors of ψ_s with respect to the transition that resets y to 0 are computed as follows.

$$\begin{aligned} \mathsf{Pre}_{\mathsf{a}}(\psi_s) &= \psi_s[y/0] \wedge \psi \\ &= 0 \leq 0 < 2 \wedge 1 < x < 2 \wedge y > x + 1 \\ &= 1 < x < 2 \wedge y > x + 1 \end{aligned}$$

b. Now, consider the assignment $x := y$ and suppose that the edge has a guard ψ given by the constraint $x > y + 1$.

$$\begin{aligned} \mathsf{Pre}_{\mathsf{b}}(\psi_s) &= \psi_s[x/y] \wedge \psi \\ &= 0 \leq y < 2 \wedge 1 < y < 2 \wedge x > y + 1 \\ &= 1 < y < 2 \wedge x > y + 1 \end{aligned}$$

□

Again, to compute $\mathsf{Pre}_e(\psi_s)$ is equivalent to visit all the regions that are e-predecessors of the regions in ψ_s, but without having to explicitly represent each one of them.

Let $s \in \mathcal{S}$ and $\psi_s \in \Psi_{\mathcal{X}}$. We denote by $\mathsf{Pre}_\varepsilon(\psi_s)$ the predicate over $\mathcal{X}$ that characterizes the set of clock valuations that can reach a clock valuation in ψ_s when the timed automaton lets time elapse at location s.

$$v \models \mathsf{Pre}_\varepsilon(\psi_s) \quad \text{iff} \quad \exists \delta \in \mathbb{R}^+ .\ v + \delta \models \psi_s \wedge \forall \delta' \in \mathbb{R}^+ .\ \delta' \leq \delta \Rightarrow v + \delta' \models I(s).$$

Property 18. $\mathsf{Pre}_\varepsilon(\psi_s) \in \Psi_{\mathcal{X}}$. □

Example 8. Consider again the example illustrated in Figure 7. Case c corresponds to letting time pass at the location.

$$\begin{aligned} \mathsf{Pre}_\varepsilon(\psi_s) &= \exists \delta \in \mathbb{R}^+ .\ \psi_s[x/x+\delta, y/y+\delta] \\ &= \exists \delta \in \mathbb{R}^+ .\ 0 \leq y+\delta < 2 \wedge 1 < x+\delta < 2 \\ &= 0 \leq y < 2 \wedge 0 \leq x < 2 \wedge y - x < 1 \wedge x - y < 2 \end{aligned}$$

□

That is, $\mathsf{Pre}_\varepsilon(\psi_s)$ characterizes the set of regions that contains the regions characterized by ψ_s and all the regions that can reach one of them by taking only ε-transitions of the region graph.

Now, we can solve the reachability problem by computing the sequence of sets of clock constraints $B_0, B_1, \cdots$ as follows:

$$\begin{aligned} B_0 &= \langle q' \rangle \\ B_{i+1} &= \biguplus_{s \in \mathcal{S}} \left(\mathsf{Pre}_\varepsilon(B_{i,s}) \ \uplus \ \biguplus_{e \in \mathcal{E}} \mathsf{Pre}_e(B_{i,s}) \right) \end{aligned}$$

Notice that $B_{i,s}$ implies $B_{i+1,s}$ for all $i \geq 0$ and $s \in \mathcal{S}$.

Property 19. Let $B = \bigcup_{i \geq 0} B_i$, $q = (s, v)$, and $q' = (s', v')$. $[q'] \in \mathsf{Reach}([q])$ iff $\langle q \rangle$ implies B_s. □

6 Analysis using bisimulation equivalences

Recall that the region graph is the quotient of the transition system of a timed automaton with respect to the region equivalence. Another way of trying to cope with the complexity of building the region graph consists in building a graph which is the quotient with respect to an equivalence such that it is smaller than the region graph but it satisfies the same properties than the region graph, and therefore it can be used instead of the latter for verification purposes. In this section we define one such equivalence and explain how to construct the quotient graph.

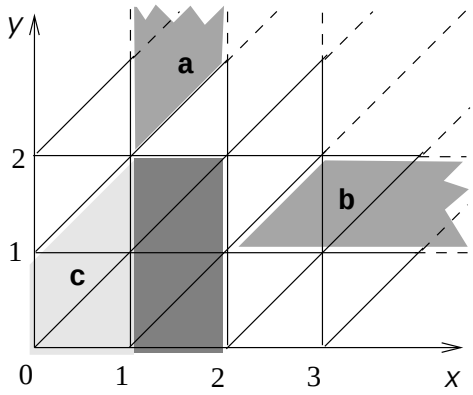


Figure 7: Computing predecessors.

6.1 Time-abstracting equivalence

A *time-abstracting bisimulation* is a *symmetric* binary relation $\mathcal{B} \subseteq \mathcal{Q} \times \mathcal{Q}$ between states such that for all $q_1, q_2 \in \mathcal{Q}$, $(q_1, q_2) \in \mathcal{B}$, if

1. For all $\sigma \in \Sigma$, whenever $q_1 \xrightarrow{\sigma} q_1'$ for some q_1', there exists q_2' such that $q_2 \xrightarrow{\sigma} q_2'$ and $(q_2, q_2') \in \mathcal{B}$.

2. For all $\delta \in \mathbb{R}^+$, whenever $q_1 \xrightarrow{\delta} q_1'$ for some q_1', there exists q_2' and $\delta' \in \mathbb{R}^+$ such that $q_2 \xrightarrow{\delta'} q_2'$ and $(q_2, q_2') \in \mathcal{B}$.

The *time-abstracting equivalence*, denoted $\approx$, is the the largest time-abstracting bisimulation. Notice that the region equivalence is a time-abstracting bisimulation. However, it is not necessarily the largest such bisimulation.

The largest time-abstracting bisimulation is an equivalence relation. Thus, computing such a bisimulation consists in constructing a partition of the state-space. Let B and C two sets of regions. We define the following operators:

$$\begin{aligned} \mathsf{Pre}_\sigma[B](C) &= \{\rho \in B \mid \exists \rho' \in C.\ \rho \xrightarrow{\sigma} \rho'\} \\ \mathsf{Pre}_\varepsilon[B](C) &= \{\rho_0 \mid \exists \rho_n \in C, n \geq 1.\ \rho_0 \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \rho_n \wedge \forall 0 \leq i < n.\ \rho_i \in B\} \end{aligned}$$

Let Ω be a partition of $\mathcal{Q}$ such that every $B \in \Omega$ is a set of regions. The quotient of $(\mathcal{Q},\ \rightarrow\ )$ with respect to Ω is transition system $(\Omega,\ \rightarrow\ )$, such that for all $B, C \in \Omega$, and for all $\ell \in \mathcal{L} = \Sigma \cup \{\varepsilon\}$,

$$B \xrightarrow{\ell} C \quad \text{iff} \quad \mathsf{Pre}_\ell[B](C) \neq \emptyset.$$

We write $B \rightarrow C$ if $B \xrightarrow{\ell} C$ for some ℓ. B is *stable* with respect to C if for all $\ell \in \mathcal{L}$, $\mathsf{Pre}_\ell[B](C)$ is either B or empty. B is stable with respect to Ω if it is stable with respect to all $C \in \Omega$. Ω is stable if all $B \in \Omega$ are stable with respect to Ω. We define $\mathsf{Suc}_\Omega(B) = \{C \in \Omega \mid B \rightarrow C\}$ and $\mathsf{Pre}_\Omega(B) = \{C \in \Omega \mid C \rightarrow B\}$.

```
Ω := Ω0;                                                          (0)
Acc := {B ∈ Ω | B ∩ Init ≠ ∅};                                    (1)
Sta := ∅;                                                         (2)
while ∃B ∈ Acc \ Sta                                              (3)
  C_B := Split(B, Ω)                                              (4)
  if (C_B = {B})                                                  (5)
  then                                                            (6)
    Sta := Sta ∪ {B};                                             (7)
    Acc := Acc ∪ Suc_Ω(B);                                        (8)
  else                                                            (9)
    Acc := Acc \ {B};                                             (10)
    Ω := (Ω \ {B}) ∪ C_B;                                         (11)
    Sta := Sta \ Pre_Ω(B);                                        (12)
    if B ∩ Init ≠ ∅                                               (13)
    then                                                          (14)
      Acc := Acc ∪ {C ∈ C_B | C ∩ Init ≠ ∅};                      (15)
end                                                               (16)
```

Figure 8: Algorithm for computing the coarsest partition.

Property 20. Let $\Omega_{\approx}$ be the partition induced by $\approx$. Clearly, $\Omega_{\approx}$ is stable.

The quotient of $(\mathcal{Q}, \rightarrow)$ with respect to the time-abstracting equivalence is the quotient of $(\mathcal{Q}, \rightarrow)$ with respect to the coarsest partition Ω which is stable.

6.2 Computing the coarsest partition

Several algorithms have been proposed in the literature to compute the coarsest partition which is stable with respect to a transition relation [35, 11]. Here we adapt the generic algorithm developed in [11] in order to construct the partition $\Omega_{\approx}$. The algorithm is illustrated in Figure 8.

The algorithm constructs the coarsest stable partition containing only classes that are reachable from a given set $Init$ of initial regions. Ω is the current partition. Acc is the set of reachable classes, that is, those classes containing at least one reachable region. $Sta \subseteq Acc$ is the set of stable reachable classes. $\mathsf{Split}(B, \Omega)$ refines the class B by choosing a class C with respect to which B is potentially unstable, and then computing $C_B = \{B_1, B_2\}$ with $B_1 = \mathsf{Pre}_\ell[B](C)$ and $B_2 = B \setminus \mathsf{Pre}_\ell[B](C)$ for some $\ell \in \mathcal{L}$. If indeed $B_i \neq \emptyset$, for $i = 1, 2$, B is *effectively* split (11-12), its predecessors become unstable (13), and the elements of C_B that contain an initial region are added to Acc (14-15). If $C_B = \{B\}$, B is both stable (7) and reachable, so its successor classes in Ω are added to the set of reachable classes (8). [5]

[5] This algorithm constructs the *minimal* finite transition system which is equivalent to

For better understanding how the algorithm works, we have explained it as if the region-graph had been constructed before. However, in order for the algorithm to be useful, it should be implemented without a-priori constructing the region-graph. This can be done following a clock-constrained based approach [41]. The operator $\mathsf{Pre}_\ell[B](C)$ can be defined as we defined the operator $\mathsf{Pre}_\ell(C)$ in Section 5. As a matter of fact, $\mathsf{Pre}_\ell[B](C)$ is defined in Appendix A since it is needed for verification of TCTL.

7 Data-structures

In this section we present two data-structures used by the implementations of the algorithms presented in the previous sections.

7.1 Difference Bound Matrices

Let $\mathcal{X} = \{x_1, \cdots, x_n\}$, and let $\Lambda \subseteq \Psi_{\mathcal{X}}$ be the set of clock constraints over $\mathcal{X}$ defined by conjunctions of constraints of the form $x_i \prec c$ and $x_i - x_j \prec c$. Let x_0 be a clock whose value is always 0, that is, its value does not increase with time as the values of the other clocks. Then, the constraints in Λ can be uniformly represented as bounds on the difference between two clock values, where for $x_i \in \mathcal{X}$, $x_i \prec c$ is expressed as $x_i - x_0 \prec c$, and $c \prec x_i$ as $x_0 - x_i \prec -c$.

Such constraints can be then encoded as a $(n+1) \times (n+1)$ square matrix D whose indices range over the interval $[0, \cdots, n]$ and whose elements belong to $\mathbb{Z} \times \{<, \leq\}$. The first column of D encodes the upper bounds of the clocks. That is, if $x_i - x_0 \prec c$ appears in the constraint, then D_{i0} is the pair $(c, \prec)$, otherwise it is $(\infty, <)$ which says that the value of clock x_i is unbounded. The first row of D encodes the lower bounds of the clocks. If $x_0 - x_i \prec -c$ appears in the constraint, D_{0i} is $(-c, \prec)$, otherwise it is $(0, \leq)$ because clocks can only take positive values. The element D_{ij} for $i, j > 0$, is the pair $(c, \prec)$ which encodes the constraint $x_i - x_j \prec c$. If a constraint on the difference between x_i and x_j does not appear in the conjunction, the element D_{ij} is set to $(\infty, <)$.

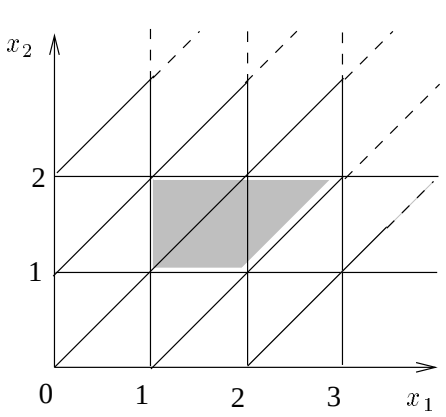
Example 9. Let Λ be the clock constraint $1 < y < 2 \wedge 1 < x \wedge x - y < 2$. Figure 9a shows its matrix representation. □

As a matter of fact, many different DBM's represent the same clock region. This is because some of the bounds may not be *tight* enough.

Example 10. Consider again the clock constraint depicted in Figure 9. The matrix b is an equivalent encoding of the clock constraint obtained by setting the upper bound of x_1 to be $(3, <)$ and the difference $x_2 - x_1$ to be $(1, <)$. Notice that this two constraints are implied by the others.

---

the region-graph upto time-abstracting bisimulation. Another different approach consists in minimizing the timed automaton itself, rather than its model, in order to obtain an equivalent timed automaton which is minimal upto a stronger notion of bisimulation. This subject has been explored in [39]. A more practical point of view is adopted in [21], also explained in Appendix B.



a

D	x_0	x_1	x_2
x_0	$(0, \leq)$	$(-1, <)$	$(-1, <)$
x_1	$(\infty, <)$	$(0, \leq)$	$(1, <)$
x_2	$(2, <)$	$(\infty, <)$	$(0, \leq)$

b

D'	x_0	x_1	x_2
x_0	$(0, \leq)$	$(-1, <)$	$(-1, <)$
x_1	$(3, <)$	$(0, \leq)$	$(1, <)$
x_2	$(2, <)$	$(1, <)$	$(0, \leq)$

Figure 9: Representation of convex sets of regions by DBM's.

However, given a clock constraint in Λ, there exists a *canonical* representative. Such a representative exists because pairs $(c, \prec)$, called *bounds*, can be ordered. This induces a natural ordering of the matrices. Bounds are ordered as follows. We take $<$ to be strictly less than $\leq$, and then $(c, \prec) \leq (c', \prec')$ iff $c < c'$ or $c = c'$ and $\prec < \prec'$. Now, $D \leq D'$ iff for all $0 \leq i, j \leq n$, $D_{ij} \leq D'_{ij}$.

Example 11. Consider the two matrices in Figure 9. Notice that $D' \leq D$.

For every clock constraint in Λ, there exists a matrix C that encodes the constraint, and for every other matrix D that also encodes the constraint, we have that $C \leq D$. Given D, we denote by $\mathsf{cf}(D)$ the matrix C.[6] We say that D is in canonical form or is a canonical representative if $D = \mathsf{cf}(D)$. Having a canonical representative gives us a simple method for checking whether two matrices represent the same constraint: D and D' encode the same constraint iff $\mathsf{cf}(D) = \mathsf{cf}(D')$.

Example 12. In Figure 9, $D' = \mathsf{cf}(D)$.

Encoding convex timing constraints by DBM's requires then $\mathcal{O}(n^2)$ memory space, where n is the number of clocks. Several algorithms have been proposed to reduce the memory space needed [21, 32]. The algorithm proposed in [21] is explained in Appendix B.

The verification algorithms require basically six operations to be implemented over matrices: conjunction, time successors, reset successors, time predecessors, reset predecessors and disjunction. These operations are implemented as follows.

Conjunction. Given D and D', $D \wedge D'$ is such that for all $0 \leq i, j \leq n$, $(D \wedge D')_{i,j} = \min(D_{ij}, D'_{ij})$.

[6] $\mathsf{cf}(D)$ can be computed using the Floyd-Warshall algorithm.

Time successors. As time elapses, clock differences remain the same, since all clocks increase at the same rate. Lower bounds do not change either since there are no decreasing clocks. Upper bounds have to be pushed to infinity, since an arbitrary period of time may pass. Thus, for a canonical representative D, $\mathsf{Suc}_\varepsilon(D)$ is such that:

$$\mathsf{Suc}_\varepsilon(D)_{ij} = \begin{cases} (\infty, <) & \text{if } j = 0, \\ D_{ij} & \text{otherwise.} \end{cases}$$

Reset successors. First notice that resetting a clock to 0 is the same as setting its value to the value of x_0, that is, $\gamma(x_i) = 0$ is the same as $\gamma(x_i) = x_0$. Now, when we set the value of x_i to the value of x_j, x_i and x_j become equal and all the constraints on x_j become also constraints on x_i. Having this in mind, the matrix characterizing the set of reset-predecessors of D by reset γ consists in just copying some rows and columns. That is, the matrix $D' = \mathsf{Suc}_\gamma(D)$ is such that for all $0 \leq i, j \leq n$, if $\gamma(x_i) = x_j$ then $\mathsf{row}_i(D') = \mathsf{row}_j(D)$ and $\mathsf{col}_i(D') = \mathsf{col}_j(D)$.

Time predecessors. To compute the time predecessors we just need to push the lower bounds to 0, provided that the matrix is in canonical form. Thus, for a canonical representative D, $\mathsf{Pre}_\varepsilon(D)$ is such that:

$$\mathsf{Pre}_\varepsilon(D)_{ij} = \begin{cases} (0, \leq) & \text{if } i = 0, \\ D_{ij} & \text{otherwise.} \end{cases}$$

Reset predecessors. Recall that the constraint characterizing the set of predecessors is obtained by substituting each clock x_i by $\gamma(x_i)$. Now suppose that we have two constraints $x_k - x_l < c_{kl}$ and $x_r - x_s < c_{rs}$ and we substitute x_k and x_r by x_i, and x_l and x_s by x_j. Then, we obtain the constraints $x_i - x_j < c_{kl}$ and $x_i - x_j < c_{rs}$ which are in conjunction, and so $x_i - x_j < \min(c_{kl}, c_{rs})$. Thus, the matrix $D' = \mathsf{Pre}_\gamma(D)$ is such that for all $0 \leq i \leq n$, $D'_{ij} = \min\{D_{kl} \mid \gamma(x_k) = x_i \wedge \gamma(x_l) = x_j\}$.

Disjunction. Clearly, the disjunction of two DBM's is not necessarily a DBM. That is, Λ is not closed under disjunction. Usually, the disjunction of D and D' is represented as the set $\{D, D'\}$. Thus, a lot of computational work is needed in order to determine whether two sets of DBM's represent the same constraint.

It may turn out, for example, that a set of DBM's can replaced by a single DBM. The tool KRONOS, for instance, use some heuristics to check whether the union of two DBM's represent is indeed a DBM. Such heuristics are applied when the number of DBM's stored in memory becomes greater than some threshold. However, checking whether a set of more than two DBM's can be represented as a DBM is computationally expensive.

7.2 Symbolic Graphs

One solution adopted in some of the verification tools to overcome the problem of the non-convexity of the union of DBM's, consists in structuring the set of reachable states as a graph rather than as a union of DBM's. The main difference between this graph and the region graph is that its nodes are pairs (s, D), where $D \in \Lambda$, instead of regions. This graph is called *forward-simulation* graph or *backward-simulation* graph, according to which method is used to construct it.

Let q and q' be two states. To verify whether $q' \in \mathsf{Reach}(q)$, the forward-simulation graph is the pair (N, E) constructed as follows:

1. $\mathsf{Suc}_\varepsilon(\langle q \rangle) \in N$.
2. For every $(s, D) \in N$, and for every $e = (s, \sigma, \psi, \gamma, s') \in \mathcal{E}$, if $D' = \mathsf{Suc}_\varepsilon(\mathsf{Suc}_e(D)) \neq \emptyset$, then $(s', D') \in N$ and $e \in E$.

The backward-simulation graph is the pair (N, E) constructed as follows:

1. $\mathsf{Pre}_\varepsilon(\langle q' \rangle) \in N$.
2. For every $(s, D) \in N$, and for every $e = (s', \sigma, \psi, \gamma, s) \in \mathcal{E}$, if $D' = \mathsf{Pre}_\varepsilon(\mathsf{Pre}_e(D)) \neq \emptyset$, then $(s', D') \in N$ and $e \in E$.

These graphs can be constructed using a depth-first or a breadth-first techniques indistinctly. For instance, the algorithm for constructing the forward-simulation graph using a breadth-first technique is illustrated in Figure 10.

These algorithms only store matrices in canonical form and only one copy of each matrix is stored, making the test whether a node already exists very simple indeed. The major drawback of these algorithms is that they can introduce a lot of redundancy, in the sense that the same region can belong to many nodes of the graph. The number of nodes of the graph can be reduced by testing whether the matrix of the newly constructed node is *included in*, instead of *equal to*, an already existing node.

Besides, they do not need to be entirely constructed, that is, their construction can be stopped as soon as a node is found to intersect $\langle q' \rangle$, in the case of the forward-simulation graph, or $\langle q \rangle$ in the other case.

7.3 Decision Diagrams

Other data-structures that have been proposed are based on the so-called *Binary Decision Diagrams*. BDD's, first introduced in [15], are efficient canonical representations of a formulas of propositional logic. BDD's have been successfully used in verification, especially in the analysis of digital circuits [16, 36].

The first idea consists in encoding the regions and the transition relation defined by the region graph using BDD's, based on the fact that the region graph is just a finite graph. This approach is followed in [7].

BDD's are also used by the algorithms based on the discretization of the state-space. For instance, when $\epsilon = 1/n$, such algorithms can be efficiently implemented using the so-called *Numeric Decision Diagrams* [8]. The idea behind

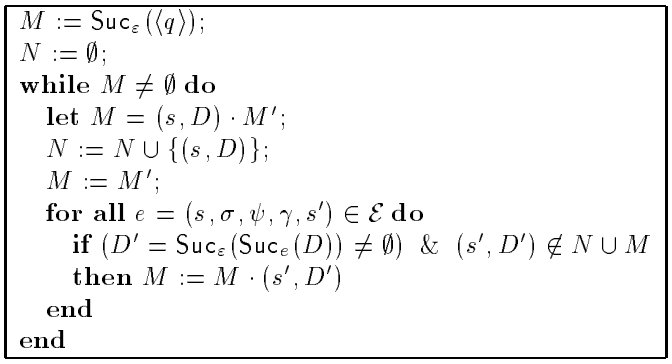


Figure 10: Forward-simulation graph using a breadth-first technique.

NDD's is very simple. Suppose that each clock can take values in the range $[0, k)$, and consider a discretization of time such that the possible clock values are $K = \{0, \ldots, k-1\}$. Each clock can be treated as a bounded integer variable and any of its possible values can be encoded in binary using $\log k$ bits. Consequently, any subset of K^n can be viewed as a subset of $\{0,1\}^{n \log k}$ and represented by a BDD over $n \log k$ boolean variables. Given a fixed variable ordering, this representation is canonical. All the operations described for DBM's can be simply implemented as operations on BDD's.

7.4 Tools

A lot of tools have been developed based on the algorithms and data-structures described in this paper. It is not the purpose of this paper to review each of them in detail. The reader is referred to the bibliography and to the web pages (whenever available) for a more detailed information about the tools.

- COSPAN [27] has been extended in several different ways to deal with timing [19, 7, 6].
- EPSILON [18][7] and REAL TIME EXPLORER [43] are two tools based on the notions of bisimulation and refinement.
- KRONOS[8] implements the model-checking algorithm for TCTL described in Appendix A, the backward and forward reachability algorithms using DBM's [20, 13] and NDD's [14], and the algorithm for computing the coarsest partition modulo the timed-abstracting bisimulation [41]. This

[7] http://www.cs.auc.dk/general/FS/epsilon-dir/folder.html
[8] http://www.imag.fr/VERIMAG/PEOPLE/Sergio.Yovine

last algorithm allows using the tool ALDEBARAN [25] [9] to check for different kinds of bisimulation equivalences.

- RT-SPIN [40] extends the on-the-fly, depth-first, forward verification algorithm of SPIN[10] using DBM's for encoding timing constraints.
- UPPAAL[11] implements forward and backward reachability algorithms based on DBM's [33, 10, 28] that allow checking for safety properties.
- VERITI [44, 23] implement a reachability algorithm that uses data-structures that combine both BDD's and DBM's to check for safety properties. The tool also implements an algorithm that computes the coarsest partition modulo the timed-abstracting equivalence [2, 1].

8 Conclusions

In this paper we have over-viewed four different techniques for analyzing properties of real-time systems specified as timed automata. For the sake of simplicity, we have only discussed how these techniques are used to solve the so-called reachability problem. Other verification problems can also be solved by adapting these four basic procedures. As an example, we discuss in Appendix A how the region-graph and clock-constraint based algorithms can be used to solve the model-checking problem for TCTL.

The algorithms we have studied always terminate with a yes or no answer for the reachability problem. As we have already mentioned in the introduction, the complexity of this problem is exponential on the number of clocks and in the encoding of the constants[12]. Even though these algorithms and the data-structures they used have been designed to try to avoid that complexity in practice, the state-space to be explored for a given system could anyway exceed the memory capabilities.

To overcome this problem, some of the algorithms described in this paper have been modified in such a way that they do no longer analyze the exact set of reachable states but an over- or under-approximated one. These algorithms rely on the fact that even if the state-space is larger, its representation is more compact, that is, it requires less memory space.

One such algorithm is the one described in [9] where the state-space is over-approximated by replacing the disjunction operator of clock constraints by the convex hull of them, that is, by the smallest convex clock constraint that contains the disjunction of them. This technique, combined with a BDD-based encoding of the state-space, permit a very compact representation of the set of reachable states. This method allows us to partially solve the verification question for safety properties. That is, the property is satisfied if the algorithm does not

[9] http://www.imag.fr/VERIMAG/DIST_SYS/aldebaran-english.html
[10] http://netlib.bell-labs.com/netlib/spin/whatispin.html
[11] http://www.docs.uu.se/docs/rtmv/uppaal/index.shtml
[12] It is indeed PSPACE-complete [4].

find an unsafe state in the larger space, however, if such a state is indeed found, we cannot immediately conclude that the system does not satisfy the property because that particular state may not belong to the exact set of reachable states.

This problem is solved in [23]. There, both under- and over-approximations of the set of reachable states are maintained. Basically, the algorithm iterates and repeatedly changes from one approximating set to the other according to the results obtained in the previous iteration. At each iteration, the approximations are made more accurate, and therefore, eventually become equal to the set of reachable states. This algorithm gives an exact answer to the problem, and it takes advantage of the compact representation of the approximated state-spaces. However, it might end up by constructing the exact state-space before being able to verify the property, in which case, it will perform worst than the exact algorithms.

References

[1] A. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proc. 13th IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, 1992.

[2] A. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. Minimization of timed transition systems. In W.R. Cleaveland, editor, *CONCUR 92: Theories of Concurrency*, pages 340–354. Lecture Notes in Computer Science 630, Springer-Verlag, 1992.

[3] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. 5th Symp. on Logics in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990. See also "Model checking in dense real time",*Information and Computation*, 104(1):2–34, 1993.

[4] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. 17th ICALP*, pages 322–335. Lecture Notes in Computer Science 443, Springer-Verlag, 1990. See also "A theory of timed automata", *Theoretical Computer Science*, 126:183–235, 1994.

[5] R. Alur and T. Henzinger. Logics and models of real-time: a survey. In *Proc. REX Workshop "Real-Time: Theory in Practice"*, the Netherlands, June 1991. Lecture Notes in Computer Science 600, Springer-Verlag.

[6] R. Alur, A. Itai, R. Kurshan, and M. Yannakakis. Timing verification by successive approximation. In *Proc. 4th Workshop on Computer-Aided Verification*. Lecture Notes in Computer Science 663, Springer-Verlag, 1992. Also in *Information and Computation*, 118(1):142–157, 1995.

[7] R. Alur and R.P. Kurshan. Timing analysis in COSPAN. In T.A. Henzinger R. Alur and E. Sontag, editors, *Hybrid Systems III*, pages 220–231. LNCS 1066, Springer-Verlag, 1996.

[8] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-structures for the verification of timed automata. In O. Maler, editor, *Proc. HART'97*, pages 346–360. LNCS 1201, Springer-Verlag, 1997.

[9] F. Balarin. Approximate reachability analysis of timed automata. In *Proc. 1996 IEEE Real-Time Systems Symposium, RTSS'96*, Washington, DC, USA, December 1996. IEEE Computer Society Press.

[10] J. Bengtsson, W. Griffioen, K. Kristorffersen, K. Larsen, F. Larsson, P. Pettersson, and Wang Yi. Verification of an audio protocol with bus using Uppaal. In *Proc. 8th Conference Computer-Aided Verification, CAV'96*, pages 244–256, Rutgers, NJ, July 1996. Lecture Notes in Computer Science 1102, Springer-Verlag.

[11] A. Bouajjani, J.C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming*, 18:247–269, 1992.

[12] A. Bouajjani, Y. Lakhnech, and S. Yovine. Model checking for extended timed temporal logics. In *Proc. 4th Intl. Symp. Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT'96*, Uppsala, Sweden, September 1996.

[13] A. Bouajjani, S. Tripakis, and S. Yovine. On-the-fly symbolic model-checking for real-time systems. In *Proc. 18th IEEE Real-Time Systems Symposium, RTSS'97*, San Francisco, USA, December 1997. IEEE Computer Society Press.

[14] M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In *Proc. 1997 Computer-Aided Verification, CAV'97*, Israel, June 1997. to appear in LNCS, Springer-Verlag.

[15] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–692, 1986.

[16] J.B. Burch, E.M. Clarke, D.Dill, L.J. Hwang, and K.L. McMillan. Symbolic model checking: 10^{20} states and beyond. In *Proc. 5th Symp. on Logics in Computer Science*, pages 428–439. IEEE Computer Society Press, 1990.

[17] K. Čerāns. Decidability of bisimulation equivalences for parallel timer processes. In *Proc. 4th Workshop on Computer-Aided Verification*. Lecture Notes in Computer Science 663, Springer-Verlag, 1992.

[18] K. Cerans, J. C. Godskesen, and K. G. Larsen. Timed modal specifications - theory and tools. In C. Courcoubetis, editor, *Proc. 5th Computer-Aided Verification*, pages 253–267. LNCS 697, Springer-Verlag, June 1993.

[19] C. Courcoubetis, D. Dill, M. Chatzaki, and P. Tsounakis. Verification with real-time COSPAN. In *Proc. 4th Workshop on Computer-Aided Verification*. Lecture Notes in Computer Science 663, Springer-Verlag, 1992.

[20] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proc. 1995 IEEE Real-Time Systems Symposium, RTSS'95*, Pisa, Italy, December 1995. IEEE Computer Society Press.

[21] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proc. 1996 IEEE Real-Time Systems Symposium, RTSS'96*, Washington, DC, USA, December 1996. IEEE Computer Society Press.

[22] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Proc. 1st Workshop on Computer-Aided Verification*, France, 1989. Lecture Notes in Computer Science 407, Springer-Verlag.

[23] D. L. Dill and H. Wong-Toi. Verification of real-time systems by successive over and under approximation. In Pierre Wolper, editor, *Proceedings of the Seventh Conference on Computer-Aided Verification, CAV'95*, Lecture Notes in Computer Science 939, pages 409–422, Liege, Belgium, 1995. Springer-Verlag.

[24] E.A. Emerson and E. Clarke. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proc. Workshop on Logic of Programs*. Lecture Notes in Computer Science 131, Springer-Verlag, 1981.

[25] J.C. Fernandez and L. Mounier. On the fly verification of behavioural equivalences and preorders. In *Proc. CAV'91*. LNCS 757, Springer-Verlag, 1991.

[26] A. Göllü, A. Puri, and P. Varaiya. Discretization of timed automata. In *Proc. 33rd CDC*, 1994.

[27] Z. Har'El and R. Kurshan. Automatic verification of coordinating systems. In J. Sifakis, editor, *Proc. 1st Workshop on Computer-Aided Verification*. Lecture Notes in Computer Science 407, Springer-Verlag, 1989.

[28] K. Havelund, A. Skou, K. G. Larsen, and K. Lund. Formal modelling and analysis of an audio/video protocol: an industrial case study using uppaal. In *Proc. 18th IEEE Real-Time Systems Symposium, RTSS'95*, San Francisco, California, USA, December 1997. IEEE Computer Society Press.

[29] T.A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proc. REX Workshop "Real-Time: Theory in Practice"*, New York, 1992. Springer-Verlag.

[30] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *Proc. 7th Symp. on Logics in Computer Science*, pages 394–406. IEEE Computer Society Press, 1992. Also in *Information and Computation*, 111(2):193–244, 1994.

[31] R. Jain. *FDDI handbook: high-speed networking using fiber and other media*. Addison-Wesley, 1994.

[32] K. G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. Efficient verification of real-time systems: compact data structure and state-space reduction. In *Proc. 18th IEEE Real-Time Systems Symposium, RTSS'95*, San Francisco, California, USA, December 1997. IEEE Computer Society Press.

[33] K. G. Larsen, P. Petterson, and Wang Yi. Compositional and symbolic model-checking of real-time systems. In *Proc. 1995 IEEE Real-Time Systems Symposium, RTSS'95*, Pisa, Italy, December 1995. IEEE Computer Society Press.

[34] K. G. Larsen and Y. Wang. Timed abstracted bisimulation: implicit specification and decidability. In *Proc. MFPS'93*, 1993.

[35] D. Lee and M. Yannakakis. Online minimization of transition systems. In *ACM Symp. on Theory of Computing*. ACM Press, 1992.

[36] K.L. McMillan. *Symbolic model-checking: an approach to the state-explosion problem*. Kluwer, 1993.

[37] M. Measche and B. Berthomieu. Time petri-nets for analyzing and verifying time dependent communication protocols. In H. Rudin and C.H. West, editors, *Protocol Specification, Testing and Verification, III*. IFIP, North-Holland, 1983.

[38] A. Olivero. Modélisation et analyse de systèmes temporisés et hybrides. Thèse, Institut National Polytechnique de Grenoble, Grenoble, France, September 1994.

[39] J.G. Springintveld and F.W. Vaandrager. Minimizable timed automata. In B. Jonsson and J. Parrow, editors, *Proc. FTRTFT'96*, Uppsala, Sweden, 1996. LNCS 1135, 130–147, Springer-Verlag.

[40] S. Tripakis and C. Courcoubetis. Extending promela and spin for real time. In *TACAS'96*, Passau, Germany, 1996. Lecture Notes in Computer Science 1055, Springer-Verlag.

[41] S. Tripakis and S. Yovine. Analysis of timed systems based on time–abstracting bisimulations. In *Proc. 8th Conference Computer-Aided Verification, CAV'96*, pages 232–243, Rutgers, NJ, July 1996. Lecture Notes in Computer Science 1102, Springer-Verlag.

[42] VERIMAG. School on Methods and Tools for the Verification of Infinite-State Systems. `http://www.imag.fr/VERIMAG`. Grenoble, France, March 1997.

[43] C. Weise and D. Lenzkes. Efficient scaling invariant checking of timed bisimulation. In *STACS'97*. Springer-Verlag, 1997.

[44] Howard Wong-Toi and David L. Dill. Approximations for verifying timing properties. In Teo Rus and Charles Rattray, editors, *Theories and Experiences for Real-Time System Development (Proceedings First AMAST Workshop on Real-Time System Development)*, chapter 7, pages 177–204. World Scientific Publishing, 1994.

[45] S. Yovine. Méthodes et outils pour la vérification symbolique de systèmes temporisés. Thèse, Institut National Polytechnique de Grenoble, Grenoble, France, May 1993.

A The logic TCTL

In the previous sections we have studied the so-called reachability problem. In this section we define the temporal logic TCTL and explain how to check whether a timed automaton satisfies a formula of the logic. TCTL is an extension of the temporal logic CTL [24].

A.1 Syntax

Let $\mathbf{A}$ be a timed automaton with set of clocks $\mathcal{X}$ and set of atomic propositions Π, and let $\mathcal{Z}$ be a set of clocks disjoint with $\mathcal{X}$, that is, $\mathcal{Z} \cap \mathcal{X} = \emptyset$. The set $\Phi_{\mathcal{X},\mathcal{Z},\Pi}$ of formulas of TCTL are defined by the following grammar:

$$\varphi \quad ::= \quad \psi \mid \pi \mid z.\varphi \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \exists\mathcal{U} \varphi_2 \mid \varphi_1 \forall\mathcal{U} \varphi_2$$

where $\psi \in \Psi_{\mathcal{X}\cup\mathcal{Z}}$, $\pi \in \Pi$, and $z \in \mathcal{Z}$.

Let $\varphi \in \Phi_{\mathcal{X},\mathcal{Z},\Pi}$. We define $\mathsf{free}(\varphi)$ to be the set of *free* clocks of φ. $\mathsf{free}(\varphi)$ is inductively defined as follows:

$$\begin{array}{lcl}
\mathsf{free}(\psi) & = & \mathsf{clk}(\psi) \\
\mathsf{free}(\pi) & = & \emptyset \\
\mathsf{free}(z.\varphi) & = & \mathsf{free}(\varphi) \setminus \{z\} \\
\mathsf{free}(\neg\varphi) & = & \mathsf{free}(\varphi) \\
\mathsf{free}(\varphi_1 \vee \varphi_2) & = & \mathsf{free}(\varphi_1) \cup \mathsf{free}(\varphi_2) \\
\mathsf{free}(\varphi_1 \exists\mathcal{U} \varphi_2) & = & \mathsf{free}(\varphi_1) \cup \mathsf{free}(\varphi_2) \\
\mathsf{free}(\varphi_1 \forall\mathcal{U} \varphi_2) & = & \mathsf{free}(\varphi_1) \cup \mathsf{free}(\varphi_2)
\end{array}$$

We say that φ is *closed* if $\mathsf{free}(\varphi) \subseteq \mathcal{X}$, that is, every occurrence of a clock $z \in \mathcal{Z}$ is under the scope of a "$z.$" operator. Table 1 shows some typical abbreviations and their intuitive meaning.

Example 13. Consider the FDDI system depicted in Figure 3. Some properties that we would like the system to satisfy are the following.

1. At any state, each station *eventually* has time to send low-priority messages. This property can be expressed in TCTL as follows:

$$(\mathtt{Idle}_1 \wedge h_1 = 0 \wedge r_1 = 0) \Rightarrow \forall\Box\forall\Diamond\, \mathtt{AT}_1.$$

Abbrev.	*Formula*	*Explanation*
$\exists\Diamond\varphi$	$true\exists\mathcal{U}\varphi$	a state satisfying φ is reachable
$\forall\Box\varphi$	$\neg\exists\Diamond\neg\varphi$	φ holds along all executions
$\forall\Diamond\varphi$	$true\forall\mathcal{U}\varphi$	all executions lead to a state satisfying φ
$\exists\Box\varphi$	$\neg\forall\Diamond\neg\varphi$	there is an execution along which φ holds everywhere
$\exists\Diamond_{\leq c}\varphi$	$z.\exists\Diamond(\varphi\wedge z\leq c)$	φ is reachable within a time less than or equal to c
$\forall\Diamond_{\leq c}\varphi$	$z.\forall\Diamond(\varphi\wedge z\leq c)$	all executions lead to φ in at most c time units

Table 1: Typical abbreviations and their meaning.

2. After having released the token, the station *eventually* gets the token in a time less than or equal to 104:

$$\mathtt{Idle}_1 \Rightarrow \forall\Diamond_{\leq 104}\mathtt{ST}_1.$$

□

A.2 Semantics

The formulas of $\Phi_{\mathcal{X},\mathcal{Z},\Pi}$ are interpreted over *extended states*. An extended state is a triplet (s,v,ζ) such that $(s,v)\in\mathcal{Q}$ is a state of $\mathbf{A}$, and $\zeta\in[\mathcal{Z}\stackrel{tot}{\to}\mathbb{R}^+]$ is a valuation of the clocks in $\mathcal{Z}$. We define $v+\zeta$ to be the following valuation:

$$(v+\zeta)(y)=\begin{cases} v(y) & \text{if } y\in\mathcal{X},\\ \zeta(y) & \text{if } y\in\mathcal{Z}.\end{cases}$$

Let $\varphi\in\Phi_{\mathcal{X},\mathcal{Z},\Pi}$ be a closed formula. An extended state (s,v,ζ) *satisfies* the formula φ, denoted $(s,v,\zeta)\models\varphi$, if

$$\begin{array}{lll}
(s,v,\zeta)\models\psi & \text{iff} & v+\zeta\models\psi\\
(s,v,\zeta)\models\pi & \text{iff} & \pi\in P(s)\\
(s,v,\zeta)\models z.\varphi & \text{iff} & (s,v,\zeta[z/0])\models\varphi\\
(s,v,\zeta)\models\neg\varphi & \text{iff} & (s,v,\zeta)\not\models\varphi\\
(s,v,\zeta)\models\varphi_1\vee\varphi_2 & \text{iff} & (s,v,\zeta)\models\varphi_1 \text{ or } (s,v,\zeta)\models\varphi_2\\
(s,v,\zeta)\models\varphi_1\exists\mathcal{U}\varphi_2 & \text{iff} & \exists r\in\mathcal{R}^\infty(s,v).\exists p\in\mathcal{P}(r).\\
 & & \quad(\Xi(p),\zeta+\Delta(p))\models\varphi_2\\
 & & \quad\wedge\\
 & & \quad\forall p'\ll p.\ (\Xi(p'),\zeta+\Delta(p'))\models\varphi_1\vee\varphi_2\\
(s,v,\zeta)\models\varphi_1\forall\mathcal{U}\varphi_2 & \text{iff} & \forall r\in\mathcal{R}^\infty(s,v).\exists p\in\mathcal{P}(r).\\
 & & \quad(\Xi(p),\zeta+\Delta(p))\models\varphi_2\\
 & & \quad\wedge\\
 & & \quad\forall p'\ll p.\ (\Xi(p'),\zeta+\Delta(p'))\models\varphi_1\vee\varphi_2
\end{array}$$

A state (s,v) satisfies φ if the extended state (s,v,ζ) satisfies φ where ζ is such that $\zeta(z)=0$ for all $z\in\mathcal{Z}$. The set of states that satisfy the formula φ, denoted $[\![\varphi]\!]$, is called the *characteristic set* of φ. $\mathbf{A}$ satisfies φ, denoted $\mathbf{A}\models\varphi$, if all the states of $\mathbf{A}$ satisfy φ.

A.3 Region-graph based algorithm

Given a timed automaton $\mathbf{A}$ and a TCTL-formula φ, we are interested in checking whether $\mathbf{A}$ satisfies φ. Let $\hat{\Psi}$ be the set of sub-formulas of φ that belong to Ψ. The region graph $\mathsf{RG}(\mathbf{A}, \hat{\Psi})$ can be used to solve the problem.

Let $[\varphi]$ to be the set of regions defined as follows.

$$\begin{array}{lcl}
[\psi] & = & \{[(s,v,\zeta)] \mid v+\zeta \models \psi\} \\
[\pi] & = & \{[(s,v,\zeta)] \mid \pi \in P(s)\} \\
[z.\varphi] & = & \{[(s,v,\zeta)] \mid [(s,v,\zeta[z/0])] \in [\varphi]\} \\
[\neg\varphi] & = & \mathcal{Q}_{\simeq} \setminus [\varphi] \\
[\varphi_1 \vee \varphi_2] & = & [\varphi_1] \cup [\varphi_2] \\
[\varphi_1 \exists\mathcal{U} \varphi_2] & = & EU([\varphi_1],[\varphi_2]) \\
[\varphi_1 \forall\mathcal{U} \varphi_2] & = & AU([\varphi_1],[\varphi_2])
\end{array}$$

where $EU(R_1,R_2) = \bigcup_{i\geq 0} E_i$ such that

$$\begin{array}{rcl}
E_0 & = & R_2 \\
E_{i+1} & = & E_i \cup R_1 \cap \mathsf{Pre}(E_i)
\end{array}$$

and $AU(R_1,R_2) = \bigcup_{i\geq 0} A_i$ such that

$$\begin{array}{rcl}
A_0 & = & R_2 \\
A_{i+1} & = & E_i \cup R_1 \cap \mathsf{Pre}(E_i) \cap \widetilde{\mathsf{Pre}}(E_i)
\end{array}$$

where $\widetilde{\mathsf{Pre}}(E_i) = \mathcal{Q}_{\simeq} \setminus \mathsf{Pre}(\mathcal{Q}_{\simeq} \setminus E_i)$. That is, a region belongs to $\widetilde{\mathsf{Pre}}(E_i)$ if all its successors belong to E_i. Thus, $\mathsf{Pre}(E_i) \cap \widetilde{\mathsf{Pre}}(E_i)$ characterizes all the regions that have a successor that belongs to E_i and all its successors belong to E_i. This prevents considering regions that do not have any successors. See Figure 11.

Property 21. Let $\mathbf{A}$ be a *Non-Zeno* timed automaton. A region ρ belongs to $EU(R_1,R_2)$ if there is a sequence of regions and transitions starting at ρ that reaches a region in R_2 such that all the intermediate regions belong to R_1:

$$EU(R_1,R_2) \quad = \quad \{\rho_0 \mid \exists \rho_0 \rightarrow \rho_1 \cdots . \exists n \in \mathbb{N}.\ \rho_n \in R_2 \wedge \forall i \leq n . \rho_i \in R_1 \cup R_2\}.$$

That is, $EU([\varphi_1],[\varphi_2])$ characterizes the set of states of $\mathbf{A}$ that satisfy the formula $\varphi_1 \exists\mathcal{U} \varphi_2$. *Non-Zenoness* ensures that the sequence of states and transitions is indeed a time-divergent execution. □

Property 22. Let $\mathbf{A}$ be a *Non-Zeno* timed automaton. A region ρ belongs to $AU(R_1,R_2)$ if every sequence of regions and transitions starting at ρ reaches a region in R_2 and all the intermediate regions belong to R_1:

$$AU(R_1,R_2) \quad = \quad \{\rho_0 \mid \forall \rho_0 \rightarrow \rho_1 \cdots . \exists n \in \mathbb{N}.\ \rho_n \in R_2 \wedge \forall i \leq n . \rho_i \in R_1 \cup R_2\}.$$

That is, $AU([\varphi_1],[\varphi_2])$ characterizes the set of states that satisfy the formula $\varphi_1 \forall\mathcal{U} \varphi_2$. This holds because in the region graph there are no self-loops labeled ε other than the ones at unbounded regions, and also because *Non-Zenoness* ensures that every region has at least one successor.

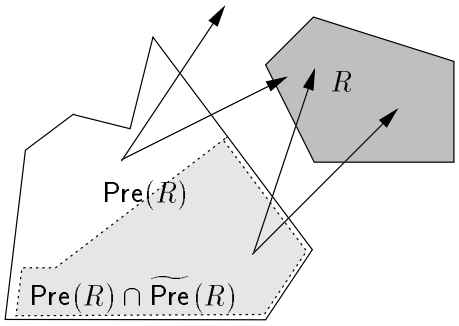


Figure 11: $\mathsf{Pre}(R)$ and $\widetilde{\mathsf{Pre}}(R)$.

Property 23. If $\mathbf{A}$ is *Non-Zeno*, $q \models \varphi$ iff $[q] \in [\varphi]$.

A.4 Clock-constraint based algorithm

Let $\mathbf{A}$ be a *Non-Zeno* timed automaton and φ a formula of TCTL. We present here an algorithm that constructs a disjoint union of clock constraints $\biguplus_{s \in \mathcal{S}} \psi_s$, denoted $(\!(\varphi)\!)$, that characterizes $[\![\varphi]\!]$ without explicitly building a-priori the region graph.

For formulas φ not containing the temporal operators $\exists\mathcal{U}$ and $\forall\mathcal{U}$, $(\!(\varphi)\!)$ is defined as follows.

$$\begin{array}{lcl}
(\!(\psi)\!) & = & \biguplus_{s \in \mathcal{S}} I(s) \wedge \psi \\
(\!(\pi)\!) & = & \biguplus_{s \in \mathcal{S}, \pi \in P(s)} I(s) \\
(\!(z.\varphi)\!) & = & \biguplus_{s \in \mathcal{S}} I(s) \wedge (\!(\varphi)\!)_s [z/0] \\
(\!(\neg\varphi)\!) & = & \biguplus_{s \in \mathcal{S}} I(s) \wedge \neg (\!(\varphi)\!)_s \\
(\!(\varphi_1 \vee \varphi_2)\!) & = & (\!(\varphi_1)\!) \cup (\!(\varphi_2)\!)
\end{array}$$

where $(\!(\varphi)\!)_s$ is the clock constraint corresponding to s in $(\!(\varphi)\!)$.

Now, let $s \in \mathcal{S}$ and $\psi', \psi_s \in \Psi_{\mathcal{X}}$. We denote by $\mathsf{Pre}_\varepsilon[\psi'](\psi_s)$ the predicate over $\mathcal{X}$ that characterizes the set of clock valuations that can reach a clock valuation in ψ_s when the timed automaton lets time elapse at location s such that all the clock valuations in between satisfy either ψ' or ψ_s.

$$\begin{array}{lll}
v \models \mathsf{Pre}_\varepsilon[\psi'](\psi_s) & \text{iff} & \exists \delta \in \mathbb{R}^+. \\
& & v + \delta \models \psi_s \\
& & \wedge \\
& & \forall \delta' \in \mathbb{R}^+.\ \delta' \leq \delta \Rightarrow v + \delta' \models I(s) \wedge \psi'.
\end{array}$$

That is, $\mathsf{Pre}_\varepsilon[\psi'](\psi_s)$ characterizes the set of regions that contains the regions characterized by ψ_s and all the regions characterized by ψ' that can reach them by taking only ε-transitions (Figure 12).

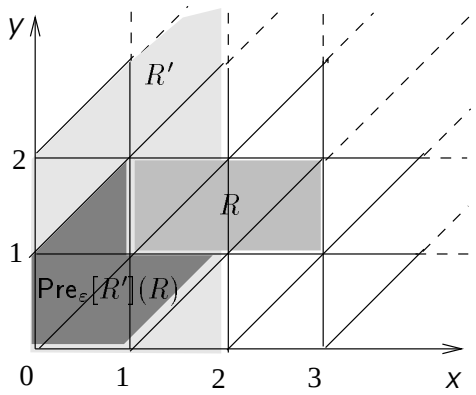


Figure 12: Conditional timed predecessors.

Property 24. $\mathsf{Pre}_\varepsilon[\psi'](\psi_s) \in \Psi_\mathcal{X}$. □

The algorithm for constructing $(\!(\varphi_1 \exists\mathcal{U} \varphi_2)\!)$ is very similar to the one based on the region graph. That is,

$$(\!(\varphi_1 \exists\mathcal{U} \varphi_2)\!) \quad = \quad EU((\!(\varphi_1)\!), (\!(\varphi_2)\!))$$

where $EU(R_1, R_2) = \bigcup_{i \geq 0} E_i$ such that

$$\begin{aligned} E_0 &= R_2 \\ E_{i+1} &= \biguplus_{s \in \mathcal{S}} \left(\mathsf{Pre}_\varepsilon[R_{1,s}](E_{i,s}) \;\uplus\; \biguplus_{e \in \mathcal{E}} \mathsf{Pre}_e(E_{i,s}) \right) \end{aligned}$$

The algorithm for computing $(\!(\varphi_1 \forall\mathcal{U} \varphi_2)\!)$ is however different to the one based on the region graph. For the sake of simplicity, we only explain here the algorithm for computing $(\!(\forall\Diamond \varphi)\!)$. The full algorithm is given in [30]. The algorithm relies on the following observation. The set of states that *eventually* reach a state in a set of regions, say A_0, can be iteratively approximated by computing the sequence of sets of regions $A_0 \subseteq A_1 \subseteq \cdots$ where $A_{i+1} = \forall\Diamond_{\leq c} A_i$. Figure 13 illustrates this observation. Now, the characteristic set of $\forall\Diamond \varphi$ is

$$(\!(\forall\Diamond \varphi)\!) \quad = \quad AD((\!(\varphi)\!))$$

where $AD(R) = \bigcup_{i \geq 0} A_i$ such that

$$\begin{aligned} A_0 &= R \\ A_{i+1} &= A_i \uplus (\!(\forall\Diamond_{\leq c} A_i)\!) \end{aligned}$$

where $c \in \mathbb{N}$ is any constant such that $c \geq 1$.

It remains now to give the algorithm to compute $(\!(\forall\Diamond_{\leq c} A_i)\!)$. This algorithm is based on the following property.

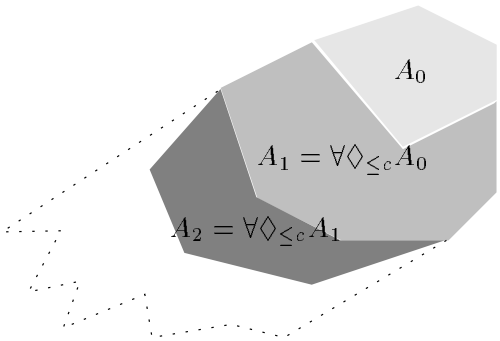


Figure 13: Computing $\forall\Diamond A_0$.

Property 25. If $\mathbf{A}$ *Non-Zeno*, $\forall\Diamond_{\leq c}\varphi$ is equivalent to $\neg z.(\neg\varphi \exists\mathcal{U} z > c)$. □

That is, all executions reach a state satisfying φ within a time less than or equal to c iff no execution continuously satisfy $\neg\varphi$ for a time greater than c. Thus,

$$(\!(\forall\Diamond_{\leq c}\varphi)\!) \quad = \quad (\!(\neg z.(\neg\varphi \exists\mathcal{U} z > c))\!).$$

A.5 Verifying the Non-Zenoness hypothesis

The two algorithms proposed above to verify whether a timed automaton $\mathbf{A}$ satisfies a TCTL-formula φ rely on the hypothesis that $\mathbf{A}$ is in fact a *Non-Zeno* timed automaton. But how do we check if $\mathbf{A}$ is indeed *Non-Zeno*?.

Recall that $\mathbf{A}$ is *Non-Zeno* if every state has a time-divergent execution starting at it. The following property has been proven in [30].

Property 26. $\mathbf{A}$ is *Non-Zeno* iff for all states $q \in \mathcal{Q}$, $q \models \exists\Diamond_{=1} true$.

In other words, verifying the *Non-Zenoness* hypothesis amounts to checking whether all the states of $\mathbf{A}$ can let time progress by 1.

Now, if $\mathbf{A}$ turns to be a *Zeno* timed automaton, we can compute the largest set of *Non-Zeno* states as $NZ = \bigcap_{i\geq 0} NZ_i$, where:

$$\begin{aligned} NZ_0 &= true \\ NZ_{i+1} &= NZ_i \cap (\!(\exists\Diamond_{=1} NZ_i)\!) \end{aligned}$$

From $\mathbf{A}$, we can construct the *Non-Zeno* timed automaton $\mathbf{A}_{NZ}$ containing all the *Non-Zeno* states of $\mathbf{A}$ by taking the invariant condition of each location $s \in \mathcal{S}$ to be NZ_s.

B Reduction of the number of clocks

Recall that the complexity of analyzing timed automata is exponential on the number of clocks. In this section we present two algorithms that can be used to reduce the number of clock variables of a timed automaton. These algorithms have been developed in [21].

B.1 Equivalence

Let $\mathbf{A}$ be a timed automaton and $(\mathcal{Q}, \rightarrow)$ its transition system.

A *timed bisimulation* $\mathcal{B} \subseteq \mathcal{Q} \times \mathcal{Q}$ is a *symmetric* binary relation between states such that for all $q_1, q_2 \in \mathcal{Q}$, $(q_1, q_2) \in \mathcal{B}$, if for all $\ell \in \Sigma \cup \mathbb{R}^+$,

1. if $q_1 \xrightarrow{\ell} q_1'$ for some q_1', there exists q_2' such that $q_2 \xrightarrow{\ell} q_2'$ and $(q_1', q_2') \in \mathcal{B}$.

The *timed equivalence*, denoted $\sim$, is the the largest timed bisimulation.

B.2 Renaming

Given a timed automaton $\mathbf{A}$ over a set of clocks $\mathcal{X}$, our aim is to find a set of clocks $\mathcal{Z}$ smaller than $\mathcal{X}$, such that appropriately renaming conditions and assignments of $\mathbf{A}$ with clocks in $\mathcal{Z}$ we obtain an equivalent behavior.

Let $\mathcal{X}$ and $\mathcal{Z}$ be two disjoint sets of clocks, $\mathbf{A}$ be a timed automaton over $\mathcal{X}$, and $\Re$ be a family of partial functions $\Re_s$ from $\mathcal{X}$ to $\mathcal{Z}$. We denote $\Re(\mathbf{A})$ the timed automaton obtained from $\mathbf{A}$ by replacing clocks in $\mathcal{X}$ by clocks in $\mathcal{Z}$ in all conditions and assignments as follows. For all $s \in \mathcal{S}$,

1. $\Re(I(s)) \in \Psi_{\mathcal{Z}}$ is obtained by replacing every $x \in \mathcal{X}$ by $\Re_s(x)$ in $I_(s)$, and
2. for every edge $e = (s, \sigma, \psi, \gamma, s) \in \mathcal{E}$, the corresponding edge $\Re(e) = (s, \sigma, \Re(\psi), \Re(\gamma), s')$ is such that,
   (a) $\Re(\psi) \in \Psi_{\mathcal{Z}}$ is obtained by replacing every $x \in \mathcal{X}$ by $\Re_s(x)$ in ψ,
   (b) $\Re(\gamma)$ is the assignment $\gamma' \in ?_{\mathcal{Z}}$ such that $\gamma' \circ \Re_{s'} = \Re_s \circ \gamma$, where $\circ$ denotes the composition of functions.

For $\Re(\mathbf{A})$ to be well defined we need to require that for all $s \in \mathcal{S}$,

1. $\Re_s(x)$ is defined for all $x \in \mathsf{clk}(I(s))$, and
2. for all $e = (s, \sigma, \psi, \gamma, s') \in \mathcal{E}$,
   (a) $\Re_s(x)$ is defined for all $x \in \mathsf{clk}(\psi)$, and
   (b) $\Re_{s'}(x) = \Re_{s'}(y)$ then $\Re_s(\gamma(x)) = \Re_s(\gamma(y))$.

The aim is to find a set of clocks $\mathcal{Z}$ with $\mathsf{card}(\mathcal{Z}) \leq \mathsf{card}(\mathcal{X})$ and clock renaming $\Re$ from $\mathcal{X}$ to $\mathcal{Z}$ such that $\Re(\mathbf{A})$ is bisimular to $\mathbf{A}$. This notion of reduction is global, that is, $\mathsf{card}(\mathcal{Z})$ of clocks are globally required to model the same behavior. However, it may happen that not all the clocks are always necessary.

More formally, it may be the case that $\Re_s$ is such that every state (s, v) is bisimular to $(s, \hat{v})$, where $\hat{v} \in \mathcal{V}_{\Re_s(\mathcal{X})}$, is such that $\hat{v}(z) = v(x)$, for $z = \Re_s(x)$. Therefore, only $\mathsf{card}(\Re_s(\mathcal{X}))$ of clocks are locally needed at location s to model the same behavior.

B.3 Activity

The first algorithm is based on the notion of *activity* of a clock. Intuitively, a clock is *active* at some location if its value at the location may influence the future evolution of the system. This may happen whenever the clock appears in the invariant condition of the location, it is tested in the condition of some of the outgoing edges, or an active clock takes its value when moving through an outgoing edge.

We define the function act that associates with each location $s \in \mathcal{S}$ the set $\mathsf{act}(s) \subseteq \mathcal{X}$ of active clocks at s as follows. For all $s \in \mathcal{S}$, $\mathsf{act}(s) = \bigcup_{i \geq 0} \mathsf{act}_i(s)$, where:

$$\begin{aligned} \mathsf{act}_0(s) &= \mathsf{clk}(I(s)) \cup \bigcup_{(s,\sigma,\psi,\gamma,s') \in \mathcal{E}} \mathsf{clk}(\psi) \\ \mathsf{act}_{i+1}(s) &= \mathsf{act}_i(s) \cup \bigcup_{(s,\sigma,\psi,\gamma,s') \in \mathcal{E}} \gamma(\mathsf{act}_i(s')) \end{aligned}$$

where $\gamma(X) \subseteq \mathcal{X}$ is such that, $y \in \gamma(X)$ iff there exists $x \in \mathcal{X}$ such that $\gamma(x) = y$.

Notice that the definition of $\mathsf{act}_{i+1}(s)$ says that if a clock x is active in s', and there is an edge with an assignment γ such that $\gamma(x) = y$, then y is active in s, since in fact, the value of x in s' is the value of y in s.

Property 27. Let $s \in \mathcal{S}$ and $v, v' \in \mathcal{V}_{\mathcal{X}}$, with $v(x) = v'(x)$ for all $x \in \mathsf{act}(s)$. Then, $(s, v) \sim (s, v')$.

Property 28. Let $\Re$ be any renaming such that for all $s \in \mathcal{S}$, $\Re_s$ is an injective function defined for all $x \in \mathsf{act}(s)$. Then, $\mathbf{A} \sim \Re(\mathbf{A})$.

B.4 Equality

The second algorithm is based on the notion of *equality* between clocks. Intuitively, two clocks $x, y \in \mathcal{X}$ are equal in $s \in \mathcal{S}$ if they have the same value in that location for every run, that is, if for every reachable state (s, v) we have that $v(x) = v(y)$. In this case, only one of the clocks is necessary to determine the behavior of the system at the location.

We define the equality relation such that two clocks are equal in a location if they are set by the assignment of every incoming edge either both to 0 or to clocks that are themselves equal in the source location.

Let $R \subseteq \mathcal{X} \times \mathcal{X}$. We denote R^* the relation $R \cup \{(0, 0)\}$. Let γ be an assignment. We denote $\gamma(R)$ the set of pairs $(x, y) \in \mathcal{X} \times \mathcal{X}$ such that $(\gamma(x), \gamma(y)) \in R^*$.

Now we formally define the equality relation $\mathtt{equ}$ to be the family of relations such that for all $s \in \mathcal{S}$, $\mathtt{equ}(s) = \bigcup_{i \geq 0} \mathtt{equ}_i(s)$, where:

$$\begin{aligned} \mathtt{equ}_0(s) &= \mathcal{X} \times \mathcal{X} \\ \mathtt{equ}_{i+1}(s) &= \mathtt{equ}_i(s) \cap \bigcap_{(s',\sigma,\psi,\gamma,s) \in \mathcal{E}} \gamma(\mathtt{equ}_i(s')) \end{aligned}$$

That is, the algorithm starts by assuming that all clocks are equal everywhere. The definition of $\mathtt{equ}_{i+1}(s)$ says that if two clocks x and y are equal in s, and they both get assigned the values of two clocks x' and y' that are equal in s' or they are both reset to 0 when taking a transition from s' to s, then they remain equal. However, if x' and y' are not equal in s', or one of them is reset but not the other, then the pair (x, y) is removed from the relation associated with s.

Property 29. For all $s \in \mathcal{S}$, $\mathtt{equ}(s)$ is an equivalence relation.

Property 30. Let $\Re$ be any renaming such that for all $s \in \mathcal{S}$, $\Re_s$ is a total function such that for all $x, y \in \mathcal{X}$, $\Re_s(x) = \Re_s(y)$ iff $(x, y) \in \mathtt{equ}(s)$. Then, $\mathbf{A} \sim \Re(\mathbf{A})$.