Minimization of Timed Transition Systems

R. Alur
AT&T Bell Laboratories
Murray Hill, New Jersey

C. Courcoubetis
University of Crete
Heraklion, Greece

N. Halbwachs
IMAG Institute
Grenoble, France

D. Dill, H. Wong-Toi
Stanford University
Stanford, California

1 Introduction

Model checking is a powerful technique for the automatic verification of finite-state systems [9, 13, 8]. A model-checking algorithm determines whether a finite-state system, represented by its state-transition graph, satisfies its specification given as a temporal logic formula. For *speed independent* or *delay insensitive* systems, the correctness can be proved by abstracting away real-time retaining only the sequencing of state-transitions. For such systems, model checking has a long history spanning over ten years, and has been shown to be useful in validating protocols and circuits [7]. Only recently there have been attempts to extend these techniques to verification of timing properties that explicitly depend upon the actual magnitudes of the delays [14, 4, 2, 1, 17, 3]. Because of the practical need for some support for developing reliable *real-time* systems, the interest in studying these techniques further is considerable. The initial theoretical results indicate that the addition of timing constraints makes the model-checking problem harder: in addition to the state-explosion problem inherent in qualitative model checking, now we also have to deal with the blow-up caused by the magnitudes of the delay bounds. Clearly, to make the proposed algorithms applicable to substantial examples there is a need to develop heuristics. In this paper, we show how to apply state-minimization techniques to verification algorithms for real-time systems.

We use *timed automata* as a representation of real-time systems [11, 2]. A timed automaton provides a way of annotating a state-transition graph of the system with timing constraints. It operates with a finite-state control and a finite number of fictitious time-measuring elements called *clocks*. Various problems have been studied in the framework of timed automata [2, 1, 3, 19, 18]. Before we can say how we improve the existing algorithms, let us recall how these algorithms work. First notice that a *state* of a timed automaton needs to record the location of the control and the (real) values for all its clocks, and thus, a timed automaton has infinitely many states. The algorithms for timed automata rely on partitioning the uncountable state space into finitely many *regions* and constructing a quotient called the *region graph*. States in the same region are in some sense equivalent, and the region graph is adequate for solving many problems. For instance, it can be used for testing emptiness of a timed automaton [2], real-time model-checking [1], testing bisimulation equivalence [19], finding bounds on the delays [10], and controller synthesis [20]. The main hurdle in implementing such algorithms using the region graph is that it's too big – it is exponential in the number of clocks and in the length of timing constraints. Recently, to overcome this problem Henzinger et al. have shown how to compute certain timing properties of timed automata symbolically [15]. We propose another approach, namely, of applying a state-minimization algorithm while constructing the region graph to reduce its size.

The objective of the minimization algorithm is to construct a minimal reachable region graph from a timed automaton. Note that we want to construct such a minimal graph without constructing the full region graph first. Recently, algorithms have been proposed for performing simultaneously the reachability analysis and minimization from an implicitly defined transition system [5, 6, 16]. First we show how these algorithms can be adapted to our needs to construct the minimal region graph. Next we extend these methods to propose an algorithm for the problem of deciding whether a timed automaton meets a specification in TCTL — a real-time extension of the branching-time logic CTL. The minimal region graph, in itself, is not adequate for checking TCTL properties. Firstly, it does not incorporate the "non-Zeno" assumption about real-time behaviors which requires that time progresses without any bound along an infinite sequence of transitions. Secondly, the minimization algorithm concerns only with reachability, and not with "timed" reachability (e.g. to check a temporal property of the form "within time 3" we need to check whether a sequence of transitions is possible within the specified bound 3). We show how to refine the minimal region graph to incorporate these requirements, and this leads to an algorithm for model checking. A nice feature of the algorithm is that it splits the minimal graph only as much as needed depending on the TCTL-formula to be checked. We remind the reader that model-checking for TCTL has been shown to be computationally hard, namely, PSPACE-complete [1]. However, examples indicate that the minimized region graph is much smaller than the worst-case exponential bound, and consequently, our methods should result in a big saving.

The rest of the paper is organized as follows. Section 2 reviews the definitions of timed automata and region graphs. In Section 3 we review the minimization algorithm, and in the following section we show how to construct the minimal region graph using it. Section 5 gives examples illustrating the construction of the minimal region graph. In the final section we consider extensions needed to do model checking for TCTL.

2 Timed automata and region graphs

In this section we recall the definition of *timed automata* and the principles of their analysis by means of finite *region graphs* [11, 2, 1].

2.1 Timed Automata

Timed automata have been proposed to model finite-state real-time systems. Each automaton has a finite set of *locations* and a finite set of *clocks* which are real-valued variables. All clocks proceed at the same rate and measure the amount of time that has elapsed since they were started (or reset). Each transition of the system might reset some of the clocks, and has an associated enabling condition which is a constraint on the values of the clocks. A transition can be taken only if the current clock values satisfy its enabling condition.

For example, the automaton of Figure 1 represents a system with four locations and two clocks x, and y. The clock x gets initialized on the transition from s_0 to s_1. At any instant, the value of x equals the time elapsed since the last time this transition was taken. The enabling condition

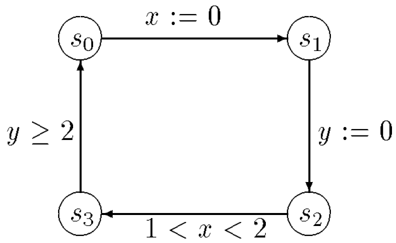

Figure 1: An example of a timed automaton

associated with the s_2 to s_3 transition expresses the following timing constraint: the delay between the transition from s_0 to s_1 and the transition from s_2 to s_3 has lower bound 1 and upper bound 2. Similarly, the clock y constrains the transition from s_3 to s_0 to occur at least two units later than the transition from s_1 to s_2. Thus to express a bound on the delay between two transitions, we reset a clock with the first transition, and associate an enabling condition with the other transition.

For each transition, the enabling condition is required to be a convex polyhedron of $\mathbb{R}^n$ ($\mathbb{R}$ denotes the set of nonnegative reals, and n is the number of clocks in the system), consisting of all the solutions of a system of linear inequalities of the form

- $x \leq k$, $x < k$, $x \geq k$, $x > k$, where x is a clock and k is an integer, or
- $x - y \leq k$, $x - y < k$, where x and y are clocks and k is an integer.

In this paper, such a polyhedron will be called a (time) *zone*. Let $\mathcal{Z}(n)$ (or simply $\mathcal{Z}$) be the set of zones of $\mathbb{R}^n$. We consider also a set of *reset actions* $\mathcal{A}(n)$ (or simply $\mathcal{A}$), which are functions from $\mathbb{R}^n$ to $\mathbb{R}^n$. For each $a \in \mathcal{A}$, there is a set of indexes $I_a \subseteq \{1 \ldots n\}$ such that

$$\forall\, \vec{x} \in \mathbb{R}^n, \forall i = 1, \ldots, n, \quad a(\vec{x})[i] = \begin{cases} 0 & \text{if } i \in I_a \\ \vec{x}[i] & \text{otherwise} \end{cases}$$

A timed automaton G is a tuple (S, C, s_{init}, T) where

1. S is a finite set of locations,
2. $C = \{x_1, \ldots, x_n\}$ is a set of clocks,
3. $s_{init} \in S$ is an initial location,
4. $T \subseteq S \times \mathcal{Z}(n) \times \mathcal{A}(n) \times S$ is a transition relation. A transition (s, z, a, s') in T will be denoted by $s \xrightarrow{z,a} s'$.

The automaton G starts with the control at the location s_{init} with all its clocks initialized to 0. The values of all the clocks increase uniformly with time. At any point in time, the automaton can make a transition, if the current values of the clocks belong to the associated zone. The transitions are instantaneous. With each transition $s \xrightarrow{z,a} s'$, the clocks in I_a get reset to 0 and start counting time with respect to that transition. At any instant, the state of the system can be fully described by specifying the current location and the values of all its clocks. So, a *state* of the system is a pair $\langle s, \vec{x} \rangle$, where $s \in S$ and $\vec{x} \in \mathrm{I\!R}^n$.

Now we can define a timed consecution relation on the states of a timed automaton. For $\delta \in \mathrm{I\!R}$, a state $\langle s', \vec{x'} \rangle$ is said to be δ-successor of another state $\langle s, \vec{x} \rangle$, written $\langle s, \vec{x} \rangle \overset{\delta}{\Rightarrow} \langle s', \vec{x'} \rangle$, iff either

- $\delta = 0$ and there is a transition $s \xrightarrow{z,a} s' \in T$ such that $\vec{x} \in z$ and $\vec{x'}$ equals $a(\vec{x})$, or
- $\delta > 0$ and $s' = s$ and $\vec{x'} = \vec{x} + \vec{\delta}$ (where $\vec{\delta}$ denotes the n-tuple $[\delta, \delta, \ldots] \in \mathrm{I\!R}^n$).

A state $\langle s', \vec{x'} \rangle$ is said to be a successor of another state $\langle s, \vec{x} \rangle$, written $\langle s, \vec{x} \rangle \Rightarrow \langle s', \vec{x'} \rangle$, iff there exists a $\delta \in \mathrm{I\!R}$ such that $\langle s, \vec{x} \rangle \overset{\delta}{\Rightarrow} \langle s', \vec{x'} \rangle$.

The behavior of a timed automaton can now be formally defined using the consecution relation $\Rightarrow$. A run of the automaton started in a state $\langle s, \vec{x} \rangle$ is obtained by iterating the relation $\Rightarrow$. Formally, a *run* r is an infinite sequence of locations $s_i \in S$, clock vectors $\vec{x_i} \in \mathrm{I\!R}^n$, and time values $\delta_i \in \mathrm{I\!R}$ of the form

$$\langle s_0, \vec{x_0} \rangle \overset{\delta_0}{\Rightarrow} \langle s_1, \vec{x_1} \rangle \overset{\delta_1}{\Rightarrow} \langle s_2, \vec{x_2} \rangle \overset{\delta_2}{\Rightarrow} \cdots \overset{\delta_{n-1}}{\Rightarrow} \langle s_n, \vec{x_n} \rangle \overset{\delta_n}{\Rightarrow} \cdots$$

Note that the above definition allows more than one transitions to occur at the same time. This means that, the time of the clocks is stopped, and the system can perform instantaneously several transitions which are enabled one after the other; each transition is enabled by the clock values which the previous one produced. Such an assumption allows the modeling of simultaneous actions of different components by interleaving; however, it is not essential for our algorithms.

The run r is called *progressive* iff the sequence of sums $\Sigma_{i=0}^{k} \delta_i$ is unbounded. This requirement corresponds to the "non-Zeno" constraint which rules out the runs in which an infinite number of transitions occur in a bounded interval of time. Thus the actual behavior of a real-time system gives rise only to progressive runs, and hence, while checking temporal properties of a timed automaton, we will restrict attention only to the progressive runs.

2.2 Region graphs

The key to solving verification problems for timed automata is construction of a finite region graph [1]. This solution constructs a specific region graph, we generalize this notion here.

A *region* $F \subseteq S \times \mathrm{I\!R}^n$ is a set of states. Typically F will be of the form $\{\langle s, \vec{x} \rangle \mid \vec{x} \in Z\}$, denoted by $\langle s, Z \rangle$, for a zone Z. For a state $\langle s, \vec{x} \rangle$ in a region F and a region F', the consecution relation $\langle s, \vec{x} \rangle \Rightarrow F'$ holds iff one of the following two conditions are met:

- *Elapse of time:* Starting from the state $\langle s, \vec{x} \rangle$, as time elapses, the state enters the region F' while staying in the region $F \cup F'$ along the way. That is, for some $\delta > 0$, $\langle s, \vec{x} + \vec{\delta} \rangle \in F'$, and the set of states $\{\langle s, \vec{x} + \vec{\delta'} \rangle \mid 0 \leq \delta' \leq \delta\}$ is entirely included in the region $F \cup F'$.
- *Eventual explicit transition:* Starting from the state $\langle s, \vec{x} \rangle$, the state stays in the region F as time elapses, and then enters F' because of an explicit transition. That is, for some $\delta \geq 0$ and some $\langle s', \vec{x'} \rangle \in F'$, the set $\{\langle s, \vec{x} + \vec{\delta'} \rangle \mid 0 \leq \delta' \leq \delta\}$ is entirely included in F, and $\langle s, \vec{x} + \vec{\delta} \rangle \overset{0}{\Rightarrow} \langle s', \vec{x'} \rangle$.

A partition R of the state space $S \times \mathbb{R}^n$ into regions is said to be *stable* iff

1. *R is stable with respect to the elapsing of time*: For every $\langle s, \vec{x} \rangle$ in F, if $\langle s, \vec{x} \rangle$ can lead to a region $F' \in R$ by letting the time elapse, then every other state $\langle s', \vec{x'} \rangle$ in F can also lead to F' by letting the time elapse.

2. *R is stable with respect to explicit transitions*: For every $\langle s, \vec{x} \rangle$ in F, if $\langle s, \vec{x} \rangle$ can lead to a region $F' \in R$ by eventually enabling an explicit transition, then every other state $\langle s', \vec{x'} \rangle$ in F can also lead to F' by eventually enabling an explicit transition (not necessarily the same transition as $\langle s, \vec{x} \rangle$).

Intuitively, stability of R means that all states in a region are equivalent with respect to the reachability analysis: if for some state $\langle s, \vec{x} \rangle \in F$, there is a state $\langle s', \vec{x'} \rangle \in F'$ such that $\langle s, \vec{x} \rangle \Rightarrow^* \langle s', \vec{x'} \rangle$, then for every state $\langle u, \vec{y} \rangle \in F$ there is a state $\langle u', \vec{y'} \rangle \in F'$ such that $\langle u, \vec{y} \rangle \Rightarrow^* \langle u', \vec{y'} \rangle$. Also our definitions ensure that the paths leading $\langle s, \vec{x} \rangle$ and $\langle u, \vec{y} \rangle$ to F' visit the same sequence of regions of R along the way. Thus, the reachability questions about the states of a timed automaton can be reduced to reachability questions about the regions of a stable partition. In general, given an initial partition of the state space, we will be interested in constructing a partition that is stable and refines the initial partition (a partition R refines another partition R' if every region F of R is entirely contained in some region F' of R'). This motivates the following definition.

A *region graph* corresponding to a timed automaton G and an initial partition R_0 of the state space of G, is a graph $RG(G, R_0) = (R, E)$ such that

1. R is a stable partition of $S \times \mathbb{R}^n$,

2. R refines the initial partition R_0, and

3. there is an edge from F to F' in E iff $\langle s, \vec{x} \rangle \Rightarrow F'$ for some state $\langle s, \vec{x} \rangle$ in F.

Clearly, we can define a region graph in which every region contains a single state. But this is not useful, because a timed automaton has infinitely many states. The following proposition, which is the main result of [2], states that it can always be folded into a finite region graph:

Proposition : For any timed automaton G and the initial partition $R_0 = \{\langle s, \mathbb{R}^n \rangle \mid s \in S\}$, there exists a *finite* region graph $RG(G, R_0)$. $\Box$

The proof of this proposition is based on the existence of the *detailed region graph $DRG(G)$* (the initial partition is assumed to contain a region $\langle s, \mathbb{R}^n \rangle$ for every location s). The constructive proof defines an equivalence relation $\cong$ on $\mathbb{R}^n$. Let c be the largest constant used in defining a zone Z used in an enabling condition of G. Then, for $\vec{x}$ and $\vec{y}$ in $\mathbb{R}^n$, define $\vec{x} \cong \vec{y}$ iff for *every* zone $Z \in \mathcal{Z}$ that is defined using integer constants not greater than c, $\vec{x} \in Z$ iff $\vec{y} \in Z$. This equivalence relation has the following properties:

- The quotient $[\mathbb{R}^n / \cong]$ is finite.

- $S \times [\mathbb{R}^n / \cong]$ is a stable partition of $S \times \mathbb{R}^n$.

Any region graph is adequate for doing a finite reachability analysis, however, as we will see later, it is not fine enough to do TCTL model-checking. On the other hand, the detailed region graph is adequate to solve the model-checking problem. The only stumbling block is its size: the number of regions of $DRG(G)$ is $O(n!|S|c^n)$.

So, the problems of interest, which will be addressed in the remainder of the paper, are

- Is it possible to symbolically build a region graph smaller than the detailed region graph?
- Is it possible to use such a reduced region graph to perform full TCTL model-checking?

3 Minimization Algorithm

Bouajjani et al [6] (see also [5]) describe a general algorithm to directly generate a minimal state graph from an implicit description (e.g., a program). Let us briefly recall this algorithm, before adapting it to the generation of region graphs.

We start from a transition system $\mathcal{S} = (S, s_0, \rightarrow)$, where S is the set of states, $s_0 \in S$ is the initial state, and $\rightarrow \subseteq S \times S$ is the transition relation. A state s is said to be accessible from s_0 if and only if $s_0 \rightarrow^* s$, where $\rightarrow^*$ denotes the reflexive-transitive closure of $\rightarrow$. For a state s and a set $X \subseteq S$, we will use the notation $s \Rightarrow X$ to denote $s \rightarrow s'$ for some $s' \in X$. Let ρ be a partition of S. A class $X \in \rho$ is said to be *stable* with respect to ρ if and only if

$$\forall Y \in \rho. [(\exists x \in X, x \Rightarrow Y) \text{ implies } (\forall x \in X, x \Rightarrow Y)].$$

A partition ρ is a *bisimulation* if and only if every class of ρ is stable with respect to ρ.

The *reduction* of $\mathcal{S}$ according to a partition ρ is the transition system $\mathcal{S}|\rho$ given by $(Acc(\rho), [s_0]_\rho, \rightarrow_\rho)$, where

- $Acc(\rho)$ is the set of classes of ρ which contain at least one state accessible from s_0;
- $[s_0]_\rho$ denotes the class of ρ which contains s_0;
- $X \rightarrow_\rho Y$ iff $x \Rightarrow Y$ for some $x \in X$.

Given a transition system $\mathcal{S}$ and an initial partition ρ_0, the algorithm described in [6] explicitly builds the transition system $\mathcal{S}|\overline{\rho}$, where $\overline{\rho}$ is the coarsest bisimulation compatible with ρ_0 (that is, every class of ρ_0 is a union of classes of $\overline{\rho}$). The termination of the algorithm requires that the bisimulation $\overline{\rho}$ must have a finite number of classes. The algorithm is given below, with the following notations:

- The function *split* "splits" a class X of a partition ρ into a minimal set of subclasses which are all stable with respect to ρ;
- For a class X of ρ, $post_\rho(X)$ denotes the set of classes of ρ which contain at least one state directly accessible from a state of X: $post_\rho(X) = \{Y \mid \exists x \in X, x \Rightarrow Y\}$.
- Conversely, $pre_\rho(X)$ denotes the set of classes of ρ which contain at least one state from which a state of X is directly accessible: $pre_\rho(X) = \{Y \mid \exists y \in Y, y \Rightarrow X\}$.

In the following algorithm, ρ is the current partition, α is the set of classes of ρ which have been found accessible from (the class of) the initial state, and σ is the set of classes of ρ which have been found stable with respect to ρ.

Minimization Algorithm:

$\rho = \rho_0; \alpha = \{[s_0]_\rho\}; \sigma = \emptyset;$
while $\alpha \neq \sigma$ do
  choose X in $\alpha \setminus \sigma$;
  let $\alpha' = \mathit{split}(X, \rho)$;
  if $\alpha' = \{X\}$ then
    $\sigma := \sigma \cup \{X\}; \quad \alpha := \alpha \cup \mathit{post}_\rho(X);$
  else
    $\alpha := \alpha \setminus \{X\};$
    if $\exists Y \in \alpha'$ such that $s_0 \in Y$ then $\alpha := \alpha \cup \{Y\}$;
    $\sigma := \sigma \setminus \mathit{pre}_\rho(X);$
    $\rho := (\rho \setminus \{X\}) \cup \alpha';$
  fi
od

4 Constructing the minimal region graph

Given a timed automaton $G = (S, C, s_{init}, T)$, we can use the algorithm of Section 3 to generate a minimal region graph. Recall that the automaton G can be viewed as a transition system over $S \times \mathbb{R}^n$ with the initial state $\langle s_{init}, \vec{0}\rangle$ and the transition relation $\Rightarrow$ (which is the union of $\overset{\delta}{\Rightarrow}$, $\delta \geq 0$). For simplicity of implementation, we require every region F to be of the form $\langle s, Z\rangle$ for a zone Z. We start with some definitions.

The *set of time predecessors* of a zone Z is

$$Z_{\swarrow} = \{\vec{y} \mid \exists \vec{x} \in Z,\ \exists \delta \in \mathbb{R},\ \vec{y} = \vec{x} + \vec{\delta}\}.$$

For zones Z and Z', $Z \setminus Z'$ is some set of disjoint zones such that the set $\{Z'\} \cup Z \setminus Z'$ forms a partition of Z, and

$$Z \sqcup Z' = \{Z \cap Z'\} \ \cup\ (Z \setminus Z') \ \cup\ (Z' \setminus Z).$$

We generalize this operator to accept any finite number of arguments: For any finite set $\{Z_1, \ldots, Z_k\}$ of zones, $\bigsqcup_{i=1}^k Z_i$ is a partition of $\bigcup_{i=1}^k Z_i$ into a set $\{Z'_1, \ldots, Z'_p\}$ of disjoint zones, such that for each $i = 1 \ldots k$, $j = 1 \ldots p$, either $Z'_j \subseteq Z_i$ or $Z'_j \cap Z_i = \emptyset$. The operator $\sqcup$ extends over regions also: $\langle s, Z\rangle \sqcup \langle s, Z'\rangle = \{\langle s, Z''\rangle \mid Z'' \in Z \sqcup Z'\}$.

In order to adapt the algorithm of Section 3 to generate a minimal region graph, we could define the "precondition" function: $\mathit{pre}(F)$ is the set of states $\langle s', \vec{x'}\rangle$ which may lead to some $\langle s, \vec{x}\rangle \in F$

either by letting the time elapse (if $s = s'$), or by an explicit transition. For a region $F = \langle s, Z\rangle$ this definition translates to:

$$pre(\langle s, Z\rangle) = \langle s, Z_{\swarrow}\rangle \cup \bigcup_{s' \xrightarrow{z,a} s} \langle s', (a^{-1}(Z) \cap z)_{\swarrow}\rangle.$$

However, such a formalization doesn't take into account the fact that one cannot reach $\langle s, Z\rangle$ from $\langle s, Z'\rangle$ without going through some zone Z'' "separating" Z' from Z. For instance, one cannot reach $\langle s, \{x \geq 2\}\rangle$ from $\langle s, \{x < 1\}\rangle$ without going through $\langle s, \{1 \leq x < 2\}\rangle$ (Recall the definition of $\langle s, \vec{x}\rangle \Rightarrow F'$ for stability of regions from Section 2). In fact, we cannot formalize the right abstraction of "time elapsing", by means of a single precondition function. Instead of looking for such a precondition, we will make precise in what case a region may directly lead to another region (following [15]), and use this notion to define the function for splitting a region into stable regions.

Let $Z \Uparrow Z'$ denote the set of $\vec{x} \in Z$ for which there exists $\delta \in \mathbb{R}$ such that $\vec{x} + \vec{\delta} \in Z'$ and $\vec{x} + \vec{\delta'} \in Z \cup Z'$ for all $0 \leq \delta' \leq \delta$. It is easy to show that $Z \Uparrow Z'$ is a zone.

Now the stability of a region can be expressed as follows. A region $\langle s, Z\rangle$ is stable with respect to another region $\langle s', Z'\rangle$ if and only if

- if $s = s'$ then $Z \Uparrow Z' \in \{Z, \emptyset\}$, and
- for every transition $s \xrightarrow{z,a} s'$ [1],
  - either $a(Z \cap z) \cap Z' = \emptyset$ (this includes the case where $Z \cap z = \emptyset$),
  - or $a(Z \cap z) \subseteq Z'$ and $Z \Uparrow (Z \cap z)$ equals Z.

From this definition, we derive the function *split*: For any locations s, s' ($s \neq s'$), for any zones Z, Z',

$$split(\langle s, Z\rangle, \langle s', Z'\rangle) = \langle s, Z\rangle \sqcup \bigsqcup_{s \xrightarrow{z,a} s'} \langle s, Z \Uparrow (Z \cap z \cap a^{-1}(Z'))\rangle$$

$$split(\langle s, Z\rangle, \langle s, Z'\rangle) = \langle s, Z\rangle \sqcup \langle s, Z \Uparrow Z'\rangle \sqcup \bigsqcup_{s \xrightarrow{z,a} s} \langle s, Z \Uparrow (Z \cap z \cap a^{-1}(Z'))\rangle$$

Now all the definitions needed for applying the algorithm can be given. Let ρ be any partition of the states into regions, and let $\langle s, Z\rangle$ be a region. Then,

$$pre_{\rho}(\langle s, Z\rangle) = \{\langle s, Z'\rangle \in \rho \mid Z' \Uparrow Z \neq \emptyset\} \cup \bigcup_{s' \xrightarrow{z,a} s} \{\langle s', Z'\rangle \in \rho \mid a(Z' \cap z) \cap Z \neq \emptyset\},$$

$$post_{\rho}(\langle s, Z\rangle) = \{\langle s, Z'\rangle \in \rho \mid Z \Uparrow Z' \neq \emptyset\} \cup \bigcup_{s \xrightarrow{z,a} s'} \{\langle s', Z'\rangle \in \rho \mid a(Z \cap z) \cap Z' \neq \emptyset\},$$

$$split(\langle s, Z\rangle, \rho) = \bigsqcup_{\langle s', Z'\rangle \in \rho} split(\langle s, Z\rangle, \langle s', Z'\rangle).$$

To implement the algorithm, we simply need efficient ways for representing zones and computing simple operations on them such as $Z \sqcup Z'$, $Z \Uparrow Z'$, $a(Z)$, and $a^{-1}(Z)$.

[1] and this includes the case where $s = s'$ and there is a looping transition on s.

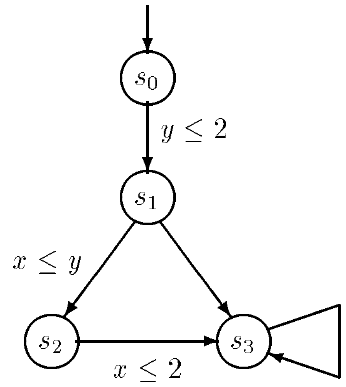

Figure 2: The timed automaton of Example 1

5 Examples

We will demonstrate the effectiveness of minimization procedure on simple examples.

5.1 Example 1

We consider first the very simple timed automaton shown on Fig. 2.

We start with an initial partition which only distinguishes regions according to their node component: $\rho = \rho_0 = \{C_0, C_1, C_2, C_3\}$, with $C_i = \langle s_i, \mathbb{R}^2 \rangle$ for $i = 0, 1, 2, 3$. Since the initial state belongs to C_0, we have $\alpha = \{C_0\}, \sigma = \emptyset$.

So, we consider first $X = C_0$. Obviously $\mathit{split}(C_0, C_2) = \mathit{split}(C_0, C_3) = \{C_0\}$, since there is no transition from s_0 to s_2 or s_3. So, $\mathit{split}(C_0, \rho) = \mathit{split}(C_0, C_1) = \{C_{00}, C_{01}\}$, with

$$C_{00} = \langle s_0, \{y \leq 2\} \rangle \quad C_{01} = \langle s_0, \{y > 2\} \rangle$$

The initial state $\langle s_0, \{x = y = 0\} \rangle$ belongs to C_{00}, so α is updated to $\{C_{00}\}$. Considering $X = C_{00}$, we find it stable with respect to $\rho = \{C_{00}, C_{01}, C_1, C_2, C_3\}$, since all of its elements can lead to C_{01} and to C_1. So, we get $\alpha = \{C_{00}, C_{01}, C_1\}$ and $\sigma = \{C_{00}\}$.

The region $X = C_{01}$ is stable with respect to ρ, and it doesn't lead to any other region. Considering $X = C_1$, we find

$$\mathit{split}(C_1, C_{00}) = \mathit{split}(C_1, C_{01}) = \mathit{split}(C_1, C_3) = \{C_1\}$$

so

$$\mathit{split}(C_1, \rho) = \mathit{split}(C_1, C_2) = \{C_{10}, C_{11}\}$$

with

$$C_{10} = \langle s_1, \{x \leq y\} \rangle \quad C_{11} = \langle s_1, \{x > y\} \rangle$$

Splitting C_1 questions about the stability of C_{00}, which is removed from σ, and considered again.

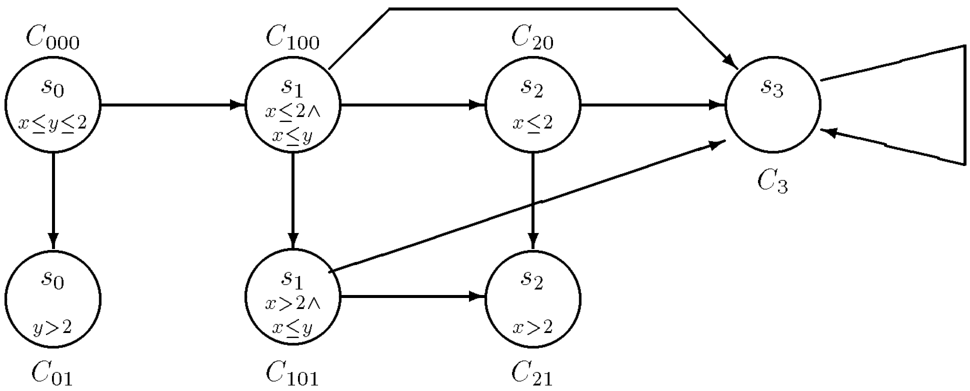

Figure 3: The minimal region graph of Example 1

Now, we have $\rho = \{C_{00}, C_{01}, C_{10}, C_{11}, C_2, C_3\}$, $\alpha = \{C_{00}, C_{01}\}$, $\sigma = \{C_{01}\}$ and $X = C_{00}$. We get $split(C_{00}, \rho) = split(C_{00}, C_{10}) = split(C_{00}, C_{11}) = \{C_{000}, C_{001}\}$, with

$$C_{000} = \langle s_0, \{x \leq y \leq 2\}\rangle \quad C_{001} = \langle s_0, \{y \leq 2 \wedge y < x\}\rangle$$

The initial state belongs to C_{000} which is stable, and can lead either to C_{01} or to C_{10}. We get $\rho = \{C_{000}, C_{001}, C_{01}, C_{10}, C_{11}, C_2, C_3\}$, $\alpha = \{C_{000}, C_{01}, C_{10}\}$, $\sigma = \{C_{000}, C_{01}\}$.

$X = C_{10}$ is found stable with respect to ρ, leading to C_2 and C_3, which become both accessible. $X = C_2$ is split into

$$C_{20} = \langle s_2, \{x \leq 2\}\rangle \quad C_{21} = \langle s_2, \{x > 2\}\rangle$$

so C_{10} must be considered again. It is split into

$$C_{100} = \langle s_1, \{x \leq y \wedge x \leq 2\}\rangle \quad C_{101} = \langle s_1, \{2 < x \leq y\}\rangle$$

C_{000} is also considered again, it is found stable and leads to C_{100}.

We have $\rho = \{C_{000}, C_{001}, C_{100}, C_{101}, C_{20}, C_{21}, C_3\}$, $\alpha = \{C_{000}, C_{01}, C_{100}\}$ and $\sigma = \{C_{000}, C_{01}\}$. Now, C_{100} is found stable, leading to C_{101}, C_{20} and C_3. C_{101} is stable and leads to C_{21} and to C_3. C_{20} is stable, and leads to C_{21} and C_3. C_{21} and C_3 are stable. The resulting graph is shown on Fig. 3. Notice that the detailed region graph of this example has 160 regions, 24 of which are accessible.

5.2 Example 2

Let us slightly complexify our example as shown by Fig. 4. The first steps of the algorithm are similar, but now C_{101} can lead to C_{001} which becomes accessible. C_{001} is found stable, leading to C_{11}. C_{11} is stable and leads to C_3. Our reduced graph has 9 accessible regions, instead of 40 in the detailed graph.

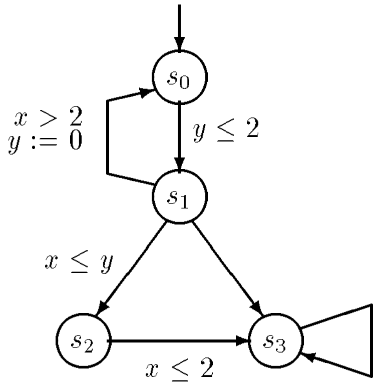

Figure 4: The timed automaton of Example 2

6 Model checking for TCTL

In this section we show how to use the algorithm for constructing the minimal region graph to check properties specified in the branching-time logic TCTL.

6.1 The logic TCTL

Let us briefly review the logic TCTL of [1]. It is a real-time extension of the branching-time logic CTL of [12]. The syntax of TCTL allows putting subscripts on the temporal operators of CTL to restrict their scope in time. Thus one can write $\exists\Diamond_{<3}\, p$ meaning "along some run within time 3." Formally, let AP be a set of atomic propositions, then the formulas ϕ of TCTL are defined inductively as:

$$\phi \;:=\; p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists\, \phi_1 \mathcal{U}_{\sim c} \phi_2 \mid \forall\, \phi_1 \mathcal{U}_{\sim c} \phi_2,$$

where p is in AP and c is an integer and $\sim$ stands for one of the binary relations $<$,$>$, $=$, $\leq$, $\geq$.

Informally, $\exists\, \phi_1 \mathcal{U}_{<c} \phi_2$ means that for some run, there exists an initial prefix of time length less than c such that ϕ_2 holds at the last state of the prefix, and ϕ_1 holds at all its intermediate states. Similarly, $\forall\, \phi_1 \mathcal{U}_{<c} \phi_2$ means that for every run, there is an initial prefix with the above property. Formally, the semantics of TCTL is defined with respect to *continuous computation trees*, but for our purposes it suffices to interpret TCTL formulas over timed automata. To interpret TCTL formulas over a timed automaton, first we need to know which atomic propositions are true in every location of the automaton. A *labeled timed automaton* is a pair (G, μ), where G is a timed automaton and μ is a labeling function from the locations of G to 2^{AP}.

Given a labeled timed automaton (G, μ), we define the satisfaction relation $\langle s, \vec{x} \rangle \models \phi$ inductively as follows:

$\langle s, \vec{x} \rangle \models p$ iff $p \in \mu(s)$.
$\langle s, \vec{x} \rangle \models \neg\phi$ iff $\langle s, \vec{x} \rangle \not\models \phi$.
$\langle s, \vec{x} \rangle \models \phi_1 \wedge \phi_2$ iff both $\langle s, \vec{x} \rangle \models \phi_1$ and $\langle s, \vec{x} \rangle \models \phi_2$.
$\langle s, \vec{x} \rangle \models \exists\, \phi_1 \mathcal{U}_{\sim c} \phi_2$ iff for some progressive run r of G starting at $\langle s, \vec{x} \rangle$, $r \models \phi_1 \mathcal{U}_{\sim c} \phi_2$.
$\langle s, \vec{x} \rangle \models \forall\, \phi_1 \mathcal{U}_{\sim c} \phi_2$ iff for every progressive run r of G starting at $\langle s, \vec{x} \rangle$, $r \models \phi_1 \mathcal{U}_{\sim c} \phi_2$.

For a run $r = \langle s_0, \vec{x_0}\rangle \stackrel{\delta_0}{\Rightarrow} \langle s_1, \vec{x_1}\rangle \stackrel{\delta_1}{\Rightarrow} \cdots$, the relation $r \models \phi_1 \mathcal{U}_{\sim c}\phi_2$ holds iff there exists k and $\delta \le \delta_k$ such that (1) $(\delta + \Sigma_{i<k}\delta_i) \sim c$, and (2) $\langle s_k, \vec{x_k}\rangle \models \phi_2$, and (3) for all $0 \le i < k$, for all $0 \le \delta' < \delta_i$, $\langle s_i, \vec{x_i} + \vec{\delta'}\rangle \models \phi_1$, and (4) for all $0 \le \delta' < \delta$, $\langle s_k, \vec{x_k} + \vec{\delta'}\rangle \models \phi_1$.

A labeled timed automaton (G, μ) satisfies a TCTL-formula ϕ iff $\langle s_{init}, \vec{0}\rangle \models \phi$. The model-checking problem for TCTL is to decide if (G, μ) satisfies ϕ. The problem is known to be PSPACE-complete [1].

6.2 Model checking algorithm

We sketch how to adapt the minimization algorithm to do model-checking for TCTL. Let (G, μ) be the labeled timed automaton with state space $S \times \mathbb{R}^n$. For a TCTL-formula ϕ, let F_ϕ be the set of states $\langle s, \vec{x}\rangle$ such that $\langle s, \vec{x}\rangle \models \phi$. The detailed region graph of [1] is adequate for TCTL model-checking: for any TCTL-formula ϕ, the set F_ϕ is a union of regions of the detailed region graph. Now our objective is to construct the set F_ϕ through only a "minimal" splitting. In our analysis, the set F_ϕ will always be a union of regions of the form $\langle s, Z\rangle$ for $Z \in \mathcal{Z}$.

The construction of F_ϕ is defined inductively on the structure of ϕ. The cases when ϕ is an atomic proposition, or is a boolean combination are simple:

$$\begin{aligned} F_p &= \cup_{p\in\mu(s)}\langle s, \mathbb{R}^n\rangle \\ F_{\neg\phi} &= (S \times \mathbb{R}^n) \setminus F_\phi \\ F_{\phi_1\wedge\phi_2} &= F_{\phi_1} \cap F_{\phi_2} \end{aligned}$$

The interesting case is when ϕ is a "timed until" formula. For simplicity of presentation, we only consider the case when ϕ is of the form $\exists\Diamond_{\sim c}\psi$ (that is, $\exists\, \mathit{true}\ \mathcal{U}_{\sim c}\psi$) or $\forall\Diamond_{\sim c}\psi$; the changes necessary to handle the "until" formulas should be obvious.

First consider an unbounded temporal formula $\phi = \exists\Diamond\psi$ (that is, $\exists\Diamond_{\ge 0}\psi$). Suppose we have constructed the set F_ψ. Let R_ψ be the partition of the states of G into two regions: F_ψ and $F_{\neg\psi}$. Now we run the minimization algorithm of Section 4 to construct a region graph $RG(G, R_\psi) = (R, E)$. Since R refines R_ψ, for any region F of R, either ψ holds at all states in F or $\neg\psi$ holds in all states in F. Suppose we want to determine the truth of the formula ϕ at the state $\langle s, \vec{x}\rangle$ in the region F of R. From the semantics of TCTL, it follows that ϕ holds at $\langle s, \vec{x}\rangle$ iff some state in F_ψ appears on a *progressive* run of G starting at $\langle s, \vec{x}\rangle$. Since every finite run can be extended to obtain a progressive infinite run, ϕ holds at $\langle s, \vec{x}\rangle$ iff some state in F_ψ is reachable from $\langle s, \vec{x}\rangle$. This holds precisely when a region $F' \in R$ such that $F' \subseteq F_\psi$ is reachable from F in the region graph. Thus, the desired set F_ϕ is a union of regions F for which some $F' \subseteq F_\psi$ is reachable from F in (R, E).

6.2.1 Progressiveness

Now consider the formula $\phi = \forall\Diamond\psi$. Suppose we construct the region graph $RG(G, R_\psi) = (R, E)$ as before. Now, ϕ holds at $\langle s, \vec{x}\rangle \in F$ iff some state in F_ψ appears on every *progressive* run of G starting at $\langle s, \vec{x}\rangle$. However, this is not equivalent to saying that every infinite path in the region graph, starting at F, contains some region $F' \subseteq F_\psi$. To determine the truth of ϕ we need to account for the progressiveness assumption while constructing the region graph.

From the results in [1] it follows that the progressiveness assumption can be modeled as *fairness constraints* on the detailed region graph which require that a path of $DRG(G)$ infinitely often visits certain sets of regions. In particular, these constraints require that for every clock i, the constraint $\vec{x}[i] = 0$ or $\vec{x}[i] > c_i$ holds at infinitely many regions along the path (here, c_i is the largest constant in a constraint involving x in the enabling conditions of G). We can use this fact to handle progressiveness in our reduced region graphs. For each clock i, let R_i be the partition of the states of G into three regions:

$$R_i^0 = \{\langle s, \vec{x}\rangle \mid \vec{x}[i] = 0\},\ R_i^{>c_i}\{\langle s, \vec{x}\rangle \mid \vec{x}[i] > c_i\},\ R_i^{\leq c_i}\{\langle s, \vec{x}\rangle \mid 0 < \vec{x}[i] \leq c_i\}.$$

Now, as the initial partition we choose the coarsest partition R'_ψ that refines R_ψ and also refines R_i for each clock i. The next step is to construct a region graph $RG(G, R'_\psi) = (R, E)$. An infinite path in this region graph is called *progressive* iff for every $i = 1 \ldots n$:

- it contains an infinite number of regions $F \subseteq R_i^0 \cup R_i^{>c_i}$.
- it contains an infinite number of regions $F \subseteq R_i^{\leq c_i} \cup R_i^{>c_i}$.

The set F_ϕ is now the union of regions F in the region graph such that every progressive path starting at F contains a region $F' \subseteq F_\psi$.

6.2.2 Timed Reachability

Now consider a formula $\phi = \exists\Diamond_{<3}\psi$. To compute whether ϕ holds at a state, we need to determine whether some state in F_ψ can be reached within 3 time units. The region graph constructed for the case $\phi = \exists\Diamond\psi$ has information only about reachability, but not about "timed" reachability. The timed reachability analysis can be performed by introducing an auxiliary clock x_0. The new state space is $S \times \mathbb{R}^{n+1}$, and the timed consecution relation $\stackrel{\delta}{\Rightarrow}$ on this new space is defined as before; the transitions corresponding to the elapse of time increment the value of x_0 along with the other clocks, and the transitions corresponding to the change of location do not depend upon the value of x_0 and leave x_0 unchanged. For $\vec{x} \in \mathbb{R}^n$ and $t \in \mathbb{R}$, let $[t]\vec{x}$ denote the $(n+1)$-vector that assigns t to the clock x_0 and agrees with $\vec{x}$ on the values of the remaining n clocks. Conversely, for $\vec{x} \in \mathbb{R}^{n+1}$, let $\vec{x}^n$ denote the n-vector obtained by discarding the value of the clock x_0.

To compute the value of ϕ at $\langle s, \vec{x}\rangle$ we consider the paths starting at $\langle s, [0]\vec{x}\rangle$. The value of x_0 is 0 at the beginning of the path and at later points its value reflects the elapsed time. The formula ϕ holds at $\langle s, \vec{x}\rangle$ iff there is a state $\langle u, \vec{y}\rangle$ reachable from $\langle s, [0]\vec{x}\rangle$ (in the extended state space) such that $\vec{y}[0] < 3$ and $\langle s, \vec{y}^n\rangle \in F_\psi$. To test this condition, we construct a region graph for the extended state space $S \times \mathbb{R}^{n+1}$. The initial partition needs to distinguish between the cases $x_0 = 0$, $0 < x_0 < 3$, and $x_0 \geq 3$ and also on the basis of the truth of ψ.

Let R_ψ be the partition of $S \times \mathbb{R}^{n+1}$ into two regions: $F'_\psi = \{\langle s, \vec{x}\rangle \mid \langle s, \vec{x}^n\rangle \in F_\psi\}$ and its complement. Let R_0 be the partition of the state space into three regions:

$$R_0^0 = \{\langle s, \vec{x}\rangle \mid \vec{x}[0] = 0\},\ R_0^{<3}\{\langle s, \vec{x}\rangle \mid 0 < \vec{x}[0] < 3\},\ R_0^{\geq 3}\{\langle s, \vec{x}\rangle \mid \vec{x}[0] \geq 3\}.$$

As the initial partition R'_ψ we choose the coarsest partition that refines both R_ψ and R_0 above, and build the region graph $RG(G, R'_\psi) = (R, E)$. Now, the truth of ϕ can be evaluated by a simple reachability analysis on this region graph. The set $F'_\phi \subseteq S \times \mathbb{R}^{n+1}$ is union of the regions $F \subseteq R_0^0$ of R for which there is a region $F' \subseteq F'_\psi \cap (R_0^0 \cup R_0^{<3})$ reachable from F. The set $F_\phi \subseteq S \times \mathbb{R}^n$ is simply the projection of F'_ϕ: $\{\langle s, \vec{x}^n\rangle \mid \langle s, \vec{x}\rangle \in F'_\phi\}$.

For a formula $\phi = \exists\Diamond_{\sim c}\psi$, the algorithm is the same; the initial partition now distinguishes between the cases $\vec{x}[0] = 0$ and $0 < \vec{x}[0] \sim c$ and $0 < \vec{x}[0] \not\sim c$. The analysis for $\phi = \forall\Diamond_{\sim c}\psi$ is similar; the initial partition now needs to account for the progressiveness assumption also (as in the case of $\forall\Diamond\psi$).

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the Fifth IEEE Symposium on Logic in Computer Science*, pages 414–425, 1990.

[2] R. Alur and D. Dill. Automata for modeling real-time systems. In *Automata, Languages and Programming: Proceedings of the 17th ICALP*, Lecture Notes in Computer Science 443, pages 322–335. Springer-Verlag, 1990.

[3] R. Alur, T. Feder, and T. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, pages 139–152, 1991.

[4] R. Alur and T. Henzinger. A really temporal logic. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 164–169, 1989.

[5] A. Bouajjani, J. Fernandez, and N. Halbwachs. Minimal model generation. In E. Clarke and R. Kurshan, editors, *Computer-Aided Verification, 2nd International Conference, CAV'90*, Lecture Notes in Computer Science 531, pages 197–203, 1990.

[6] A. Bouajjani, J. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming*, 1992. To appear.

[7] M. Browne, E. Clarke, D. Dill, and B. Mishra. Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, C-35(12):1035–1044, 1986.

[8] J. Burch, E. Clarke, D. Dill, L. Hwang, and K. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[9] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[10] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In *Proceedings of the Third Workshop on Computer-Aided Verification*, Lecture Notes in Computer Science 575, pages 399–409, 1991.

[11] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science 407, pages 197–212. Springer–Verlag, 1989.

[12] E. Emerson and E. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[13] E. Emerson and C. Lei. Modalities for model-checking: Branching time logic strikes back. In *Proceedings of the 12th ACM Symposium on Principles of Programming Languages*, pages 84–96, 1985.

[14] E. Emerson, A. Mok, A. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In E. Clarke and R. Kurshan, editors, *Computer-Aided Verification, 2nd International Conference, CAV'90*, Lecture Notes in Computer Science 531, pages 136–145, 1990.

[15] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *Proceedings of the Seventh IEEE Symposium on Logic in Computer Science*, pages 394–406, 1992.

[16] D. Lee and M. Yannakakis. Online minimization of transition systems. In *Proceedings of ACM Symposium of Theory of Computing*, 1992.

[17] H. Lewis. A logic of concrete time intervals. In *Proceedings of the Fifth IEEE Symposium on Logic in Computer Science*, pages 380–389, 1990.

[18] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. In *Real-Time: Theory in Practice, REX Workshop*, Lecture Notes in Computer Science 600, pages 549–572. Springer-Verlag, 1991.

[19] K. Čerāns. Decidability of bisimulation equivalence for parallel timer processes. In *Proceedings of the Fourth Workshop on Computer-Aided Verifiction*, Lecture Notes in Computer Science, 1992. To appear.

[20] H. Wong-Toi and G. Hoffmann. The control of dense real-time discrete event systems. In *Proceedings of the 30th IEEE Conference on Decision and Control*, pages 1527–1528, 1991.