January 1994

An Efficient Generation of the Timed Reachability Graph for the Analysis of Real-Time Systems

Inhye Kang
University of Pennsylvania

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/cis_reports

Recommended Citation


This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_reports/229
For more information, please contact libraryrepository@pobox.upenn.edu.
An Efficient Generation of the Timed Reachability Graph for the Analysis of Real-Time Systems

Abstract
As computers become ubiquitous, they are increasingly used in safety critical environments. Since many safety critical applications are real-time systems, automated analysis technique of real-time properties is desirable. Most widely used automated analysis techniques are based on state space exploration. Automatic analysis techniques based on state space exploration suffer from the state space explosion problem. In particular, a real-time system may have an unbounded number of states due to infinitely many possible time values. This paper presents our approach for generating a finite and efficient representation of the reachable states called a timed reachability graph for a real-time system. In this paper, a real-time system is specified using a timed automaton which is a timed extension of the well-known finite automaton. Our approach for coping with the state explosion problem is to extract timing information from states and to represent it as relative time relations between transitions. We also present an algorithm for computing the minimum and maximum time bounds between executions of two actions from a timed reachability graph to determine timing properties.

Comments
An Efficient Generation of the Timed Reachability Graph for the Analysis of Real-Time Systems

MS-CIS-94-36
DISTRIBUTED SYSTEMS LAB 79
LOGIC & COMPUTATION LAB 83

Inhye Kang
Insup Lee

University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

July 1994
An Efficient Generation of the Timed Reachability Graph for the Analysis of Real-Time Systems

Inhye Kang and Insup Lee
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389

Abstract

As computers become ubiquitous, they are increasingly used in safety critical environments. Since many safety critical applications are real-time systems, automated analysis technique of real-time properties is desirable. Most widely used automated analysis techniques are based on state space exploration. Automatic analysis techniques based on state space exploration suffer from the state space explosion problem. In particular, a real-time system may have an unbounded number of states due to infinitely many possible time values. This paper presents our approach for generating a finite and efficient representation of the reachable states called a timed reachability graph for a real-time system. In this paper, a real-time system is specified using a timed automaton which is a timed extension of the well-known finite automaton. Our approach for coping with the state explosion problem is to extract timing information from states and to represent it as relative time relations between transitions. We also present an algorithm for computing the minimum and maximum time bounds between executions of two actions from a timed reachability graph to determine timing properties.

1 Introduction

As computers become ubiquitous, they are increasingly used in safety critical environments. Typical safety critical applications are control systems, monitoring systems and communication systems. Any failure of such computer systems may cause a great financial loss, environmental disaster or even the loss of lives. The potential high cost associated with an incorrect operation of these systems has created a demand for a rigorous framework in which various design alternatives can be formally specified and rigorously analyzed and tested before implementation.

It is commonly believed that future safety critical systems will be more complex due to increased demands on their functionalities as well as the size of the problem domain. Thus, it will be difficult for one to analyze and test the correctness without computer-aided tools. One
common aspect of all safety critical systems is that they must respond under stringent real-time constraints. That is, their correctness depends not only on how concurrent components interact, but also on the time at which these interactions occur. In addition, these systems are costly to prototype, requiring careful prediction of timing properties before implementation and evaluation of design alternatives.

Although the verification problem is in general undecidable, there exist several automatic verification and analysis techniques for finite state systems. Such techniques are usually based on state space exploration. That is, they first identify a set of states that are reachable from the initial states and then analyze this set for verification. Such techniques exist for proving absence of deadlock or livelock, for proving properties expressed in propositional temporal logic or real-time logic, and for determining trace equivalence, testing preorder or bisimulation equivalence, etc.

The major weakness of the state space exploration based approach is that the size of the state space grows exponentially with the number of processes and thus creates the state space explosion problem. In addition, the approach is only applicable to systems with finite states. However, a real-time system has infinitely many states due to time domain. There have been several researches to construct the finite reachability graph from a real-time system [13, 11, 8, 2]. Real-time is modeled using discrete time model (e.g., the set of non-negative integers) or dense time model (e.g., the set of non-negative real numbers). Many of them assume discrete domain for time. However, the reachability analysis based on the discrete time model has the case of not detecting some reachable states in the real world where time is dense [1]. That is, the results (behaviors) obtained in the dense time model can differ from those in discrete time model. For real-time systems with dense time, there exist few researches on timed reachability analysis. This paper describes our approach to construct a timed reachability graph on a dense time model. Our approach is to develop a technique for generating the finite representation of the reachable states (called a timed reachability graph) for a real-time system by clustering a possibly infinite set of states that share reachability and timing properties.

Our model for a real-time system is a timed automaton introduced in [2]. In a timed automaton, the time domain is dense and various timing constraints can be expressed. It has a finite set of states (control locations) and finite set of real-valued clocks. The transitions may depend on the values of the clocks and can reset some of the clocks. The values of the clocks increase at the same speed with time. Our goal is to develop a technique to construct an efficient timed reachability graph of the timed automaton.

Timed automaton has been extensively studied for verification of real-time systems [6, 4, 3, 15, 10]. Most of the verification algorithms are based on a region graph [2], which is a reachability graph of a timed automaton. At any point in time, the global state of a timed automaton is given by the current state and the clock valuation. The state space has infinitely many global states due to time domain. A region graph is a finite representation of the state space by merging equivalent states in some sense into a region. However, the region graph suffers from state explosion [2]. Our approach to cope with the state explosion problem is to omit clock valuations from global states. We believe that, in most cases, including clock
valuations in the global states causes the state explosion. In our timed reachability graph, timing information is captured by relative time relations between times at which transitions are taken. In our experience, the size of the timed reachability graph is much smaller than the size of the region graph. In this paper, we describe an algorithm for constructing the timed reachability graph. In particular, we develop a precise notion of equivalence that we use for clustering states. We also present an algorithm for computing the following real-time property from the timed reachability graph: the minimum and maximum time bounds between executions of any two given actions. This property can be used to predict timing properties of the system.

The paper is organized as follows. In Section 2, we overview other methods related to our work. Section 3 presents the syntax and semantics of the timed automaton. In Section 4, we describe our algorithm for generating the reachability graph from a given timed automaton. Section 5 gives an algorithm for computing the minimum and maximum time bounds between two actions. In Section 6, we discuss and compare our approach with related work through a railroad crossing example. In Section 7, we conclude the paper with the current and future research issues.

2 Related Work

Reachability analysis is to construct a state-transition model of a system by generating all reachable states from the initial state. The state-transition model is called a reachability graph. Suppose a transition \( s \xrightarrow{x \leq 3, a?} s' \), where \( a? \) is a receive action through channel \( a \). If the current valuation of \( x \) is greater than 3 at state \( s \) or the synchronization counterpart cannot send a message through \( a \), then \( s' \) is unreachable. In real-time systems, a state can be unreachable due to timing constraints. Although timing constraints have different expressions in different models, the property that time increases uniformly and unboundedly is the same. Therefore, the number of global states in which information related to time is encoded can be infinite.

One of the most important problems in timed reachability analysis is to construct a finite timed reachability graph of the given system by clustering equivalent states in some sense. The domain of time is either discrete or dense. Many of real-time models \([13, 11, 8]\) follow the discrete time semantics since it is easier to handle and analyze. Their algorithms to construct a timed reachability graph generate the successors of the current state by increasing a time unit (or some units of time) at each step during construction of the timed reachable graph. For real-time systems with dense time, there exist little work on timed reachability analysis. The most successful method is proposed by Alur et al \([2]\). We now discuss and compare existing timed reachability analysis methods.

In Communicating Real-time Machines (CRSMs) \([13]\), a system consists of a set of CRSMs connected with one-to-one communication channels. Each CRSM has a finite set of data variables, control locations (called states) and transitions. Transitions consist of an enabling condition, an action, a transformation function and lower and upper time bounds. There are two
kinds of actions: communication and internal actions. The behaviors of the global system are
time-stamped traces of actions. Raju [12] gives a method to generate a reachability graph rep-
resenting the behaviors. In the reachability graph, each node consists of the current location of
each CRSM, the variable valuation, and the time spent by each CRSM in its current location.
Each edge is labeled with a set of actions executed and the time gap between nodes. Success-
sors of each node are generated according to the earliest possible time execution, i.e., maximal parallelism. The domain of each variable is restricted to be finite, and thus the number of possible variable valuations are finite. The time spent by each CRSM labeling a node can be distinguished using \((1 + c)\) different values, where \(c\) is the largest value (not including \(\infty\)) of upper bounds of transitions from its current location. The finite valuations of variables and time information result in the finiteness of the reachability graph.

Timed Transition Models (TTMs) is proposed to model real-time systems by Ostroff [11]. In TTM,
time is modeled using an external and conceptual global clock which ticks infinitely often. A TTM has variables including a special variable which represent the current location of the system. Transitions consist of an enabling condition, a transformation function and lower and upper time bounds. Unlike CRSMs, there is no concept of actions. When several TTM are composed, it follows the interleaving semantics not the maximal parallelism. In the reachability graph, each node consists of the history field as well as the current variable valuation. The history field represents the currently enabled transitions and pending transitions with the current time bounds. Each edge represents a transition which is either one of enabling transitions in the source node or the tick transition representing a unit of time passage. With a tick transition, the current time bounds of transitions in the history field is decreased by one up to zero. Therefore, there are \((c + 1)\) different values similar to CRSM, where \(c\) is the largest value appearing in the enabling conditions of the transitions. As long as a TTM has a finite number of valuations, the reachability graph is finite.

Modechart [8] is a graphical specification language for real-time systems. A Modechart
specification consists of modes that can be running in parallel or sequentially. A mode contains
at most one action executed for some amount of time (lower and upper time bounds are given).
A mode transition between sequential modes are labeled with enabling conditions over events
or lower and upper time bounds. A computation is an assignment of times to events. The
computations of a system can be given as a directed tree (called a computation tree) whose
paths correspond to the computations. In the computation tree, each node represents an event occurrence not a control location (mode) and each edge represents a causality. The timing information is maintained using a weighted graph called a separation graph whose weights represent lower and upper time bounds between nodes. For a finite representation of the computation tree, the computation graph is generated by collapsing the equivalent nodes in the tree. Here, the meaning of equivalence of two nodes is that the computation trees from the nodes have the same structure. The resulting computation graph is finite since there are finitely many distinguishable nodes in a computation tree. A difficulty of constructing the computation graph is that it is required to put deadlines not explicitly specified in the specification. As an example, if a successor can occur zero or more time after the current node, the successor is
divided into two nodes: one node with deadline 0 and another with delay 1 due to broadcasting communication in which a transition of a node can be triggered by events executed by its predecessors with zero time distance [14].

A timed automaton introduced in [2] is a timed extension of the well-known finite automaton. It has a finite set of state controls and finite set of real-valued clocks. The transitions may depend on the values of the clocks and can reset some of the clocks. The values of the clocks increase at the same speed with time. Timed automata have dense time semantics unlike CRSM, TTM, and Modechart. Because there can be an arbitrary number of clock variables and transitions can reset any subset of clock variables, time dependent behaviors of a real-time system are expressive. In a timed automaton, at any point in time the global state can be described by the current state and clock valuation. The system has infinitely many global states due to time domain. Alur et al. [2] provides the equivalence relation over clock valuations. Two valuations are equivalent if the integral parts of each clock are same and the orderings of fractional parts of all the clocks are same. Alur et al. construct a finite reachability graph called a region graph by merging the equivalent global states into a region. The region graph has size exponential in the number of clocks and the size of the constants that appear in the enabling conditions of the transitions [2]. In [4], regions having the same reachability are clustered and the resultant graph is called a minimal region graph. But, even the minimal region graph has exponential size.

In our approach, the timed reachability graph consists nodes corresponding to reachable states and edges corresponding to transitions in a timed automaton. Clock valuations are ignored and the relative time relations between edges (i.e., transitions) are augmented to capture the timing information. Testing the reachability of a state while generating the graph can be done by testing the satisfiability of all relations in the path up to the state. We develop a notion of equivalence of nodes such that two nodes are equivalent if they share reachable states and timing properties. The finiteness of the timed reachability graph comes from the finiteness of sets of equivalent nodes. An advantage of our approach is that the size of resultant timed reachability graph is much smaller than a region graph (even when compared to a minimal region graph). Moreover, the explicit timing relations gives a straightforward way to construct of the timed reachability graph and the power to analyze the timing properties directly from the graph. In a region graph, a region loses time information by encoding time information into a state and merging states into the region. In [2], to keep track of a desirable timing information, a new clock is introduced and then the region graph is refined with respect to the clock. Courcoubetis [6] give an algorithm with respect to a region graph to compute a timing gap between two regions which is given as the function of $\epsilon$ for $\epsilon \ll 1$, that is, an imprecise value with deviation $\epsilon$. 
3 Timed Automata

A timed automaton [2, 3, 4] has a finite set of clocks to express timing constraints in a real-time system. The values of all the clocks increase uniformly at a state and can be reset to zero on a transition. A transition can be taken if the current values of clocks satisfy the enabling condition. On the transition, its associated action is executed and its associated clocks are reset. For example, a timed automaton in Figure 1 represents a system with clocks $x$ and $y$. The system starts at the initial state $s_0$. The values of clocks $x$ and $y$ are initially zero and increase at the same speed. The clock $x$ is reset on transition $\tau_1$. At any instant, the value of $x$ equals the time elapsed since the last time $\tau_1$ was taken. Thus, $\tau_1$ can be taken at least 4 seconds after either the start of the system or the last execution of $\tau_1$. On the other hand, since the clock $y$ is never reset with any transition, $\tau_2$ can be executed only within 6 seconds after the system started.

Let $\mathbf{N}$ be the set of non-negative integers, and $\mathbf{R}$ be the set of non-negative real numbers. For simplicity, we restrict an enabling condition as a conjunction of $x \ast c$ for $x \in X$ and $\ast$ to be $\leq$ or $\geq$. We use the definition of a time automaton in [3].

**Definition 3.1** A timed automaton $M$ is a tuple $(S, s_{\text{init}}, X, \Sigma, T)$, where

1. $S$ is a finite set of states (control locations),
2. $s_{\text{init}}$ is the initial state,
3. $X$ is a finite ordered set of clocks,
4. $\Sigma$ is a finite set of actions,
5. $T \subseteq S \times Ec \times 2^X \times \Sigma \times S$ is a transition relation, where $Ec$ is the set of enabling conditions built using the boolean connectives over the atomic formulas of the form $x \ast c$ for $x \in X$ and $c \in \mathbf{N}$.

We define the following functions for convenience:

- $\pi_1 : T \rightarrow Ec$ is the projection of a transition to its enabling condition.
- $\pi_2 : T \rightarrow 2^X$ is the projection of a transition to the set of clocks that are reset.

![Figure 1: A simple timed automaton](image-url)
\begin{itemize}
  \item $\pi_3 : T \rightarrow \Sigma$ is the projection of a transition to the action.
\end{itemize}

For a transition $\tau \in T$, if the current state satisfies enabling condition $\pi_1(\tau)$, then the system may take the transition. On the transition, the system resets all clocks in $\pi_2(\tau)$ to zero, performs action $\pi_3(\tau)$, and moves to the next state, instantaneously.

Let $\vec{x} \in \mathbb{R}^k$ represent a valuation of clocks and let $\vec{\delta}$ represent a tuple $(\delta, \cdots, \delta) \in \mathbb{R}^k$ for $\delta \in \mathbb{R}$, where $k$ is the number of clock variables. The formal semantics of a timed automaton is given by executions as follows:

**Definition 3.2**

An execution of a timed automaton $M = (S, s_{\text{init}}, X, \Sigma, T)$ is defined as a finite or infinite sequence:

$$(s_0, \vec{x}_0, t_0) \xrightarrow{\tau_1} (s_1, \vec{x}_1, t_1) \xrightarrow{\tau_2} (s_2, \vec{x}_2, t_2) \cdots$$

satisfying the following properties:

- **Initialization:** $s_0 = s_{\text{init}}, \vec{x}_0 = \vec{0}, t_0 = 0$
- **Succession:** for all $i$, let $\delta_{i-1} = t_i - t_{i-1}$. Then
  - $(\vec{x}_{i-1} + \vec{\delta}_{i-1})$ satisfies $\pi_1(\tau_i)$ and
  - $\vec{x}_i(x) = \begin{cases} 
0 & \text{if } x \in \pi_2(\tau_i) \\
(\vec{x}_{i-1} + \vec{\delta}_{i-1})(x) & \text{otherwise}
\end{cases}$

Note that $t_i$ represents the time when the system control moves from $s_{i-1}$ to $s_i$, that is, the time when transition $\tau_i$ is taken.

When we analyze a system, we are usually interested in (observable) behaviors, not in the valuations of clocks.

**Definition 3.3**

A behavior of $M$ is a sequence $\langle (a_1, t_1), (a_2, t_2), \ldots \rangle$ such that there exists an execution $(s_0, \vec{x}_0, t_0) \xrightarrow{\tau_1} (s_1, \vec{x}_1, t_1) \xrightarrow{\tau_2} (s_2, \vec{x}_2, t_2) \cdots$ in $M$ and $a_1 = \pi_3(\tau_1), a_2 = \pi_3(\tau_2), \ldots$

In Figure 1, a finite sequence

$$(s_0, (x = 0, y = 0), 0) \xrightarrow{\tau_1} (s_1, (x = 0, y = 4.5), 4.5) \xrightarrow{\tau_2} (s_2, (x = 1.5, y = 6), 6)$$

is an execution of the automaton and its behavior is $\langle (a, 4.5), (b, 5), (c, 6) \rangle$. An infinite behavior

$\langle (a, 5.8), (b, 7.1), (a, 11.5), (b, 20.5), (a, 21.8), (b, 22.5), \ldots \rangle$

can be obtained from an execution,

$$(s_0, (x = 0, y = 0), 0) \xrightarrow{\tau_1} (s_1, (x = 0, y = 5.8), 5.8) \xrightarrow{\tau_2} (s_0, (x = 1.3, y = 7.1), 7.1) \xrightarrow{\tau_1} (s_1, (x = 0, y = 11.5), 11.5) \xrightarrow{\tau_2} (s_0, (x = 9, y = 20.5), 20.5) \xrightarrow{\tau_1} (s_1, (x = 0, y = 21.8), 21.8) \xrightarrow{\tau_2} (s_0, (x = 0.7, y = 22.5), 22.5) \cdots$$

does not terminate. The automaton stops.
4 Timed Reachability Graph Generation

In this section, we present how to generate the timed reachability graph, on which our analysis algorithm is applied. First, we introduce a timed reachability tree whose paths correspond to executions of a timed automaton and develop an equivalence relation between nodes. We then give an algorithm that computes the timed reachability graph by clustering equivalent nodes in the reachability tree.

4.1 The Timed Reachability Tree

The executions of a timed automaton can be given as a directed tree, called a reachability tree, in which nodes correspond to states and edges correspond to transitions. To capture timing relations, edges are augmented by timing constraints given by clocks in the timed automaton. As an example, we consider the timed automaton in Figure 1. The corresponding reachability tree is shown in Figure 2(a). For an edge e, let \( \tau(e) \) denote the variable representing the time when the system executes e (precisely, the corresponding transition of e). The timing information of the timed automaton is given as the relative time relation between edges: e.g., \( \tau(e_1) \geq \tau(e_0) + 4 \) means that the system executes e_1 at least 4 time units after it executes e_0. Since two transitions \( \tau_1 \) and \( \tau_2 \) exist from state \( s_0 \), two outgoing edges may be possible from \( n_0, n_3, n_6 \), and so on. The enabling condition of \( \tau_1 \) is \( x \geq 4 \) and x is reset on the execution of e_1, e_4, e_7, and so on. Let's consider the node \( n_6 \). Here, the system can execute e_7 at least 4 time units after \( \tau(e_4) \), that is, the timing relation of e_7 is given as \( \tau(e_7) \geq \tau(e_4) + 4 \). On the other hand, the enabling condition of \( \tau_2 \) is \( y \leq 6 \) and \( y \) is reset on \( e_0 \). But, the relation \( \tau(e_8) \leq \tau(e_0) + 6 \) cannot be satisfied since the system enters at node \( n_6 \) at least 8 time units after it enters \( n_0 \) due to edges e_1 and e_4. Thus, node \( n_8 \) is unreachable and is not included in the tree.

Before we define a reachability tree formally, we introduce some notations. For a transition \( \tau \in T \), source(\( \tau \)) and target(\( \tau \)) denote the source and target state of the transition, respectively. Similarly, for an edge e, source(e) and target(e) denote the source and target node of the edge, respectively. For an edge e, Reach(e) denotes the sequence of edges in the path from the root up to e. In Figure 2(a), Reach(e_7) = \( e_0 e_1 e_3 e_4 e_6 e_7 \). Suppose a transition \( \tau \) has an enabling condition \( \wedge_{1 \leq i \leq l}(x_i \geq c_i) \) for \( l \in N \), \( x_i \in X \) and \( c_i \in N \). For an edge e with \( \mu_2(e) = \tau \) and a sequence seq = e_0 e_1...e_l of edges, time.rel(e, seq) = \( \wedge_{1 \leq i \leq l}(\tau(e) \geq \tau(e_{x_i}) + c_i) \) such that \( e_{x_i} \) is the last edge in seq that reset \( x_i \). That is, time.rel transforms timing constraints over clocks in the timed automaton to relative time relation between edges in the reachability tree. In Figure 2(a), time.rel(e_4, e_0 e_1 e_3) is \( \tau(e_4) \geq \tau(e_1) + 4 \) because the timing constraint related to e_4, i.e., \( \pi_3(\mu_2(e_4)) \), is \( x \geq 4 \) and \( x \) is reset on transition \( \mu_2(e_1) \) not on transition \( \mu_2(e_3) \).

We now define a reachability tree. We assume \( \tau_{init} \) with \( \pi_2(\tau_{init}) := \{x \mid x \in X \} \).

**Definition 4.1** For a timed automaton \( M = (S, s_{init}, X, \Sigma, T) \), the corresponding reachability tree is a directed tree \( G = (N, n_{init}, E, \mu_1, \mu_2, \mu_3) \), where

1. \( N \) is a set of nodes,
Figure 2: A Timed Reachability Tree and Graph
2. \( n_{\text{init}} \) is the root with \( \mu_1(n_{\text{init}}) = s_{\text{init}} \),

3. \( e_{\text{init}} \) is the initial edge with \( \mu_2(e_{\text{init}}) = \tau_{\text{init}} \) and \( \mu_3(e_{\text{init}}) = \text{true} \),

4. \( E \subseteq N \times T \times N \) is a set of edges,

5. \( \mu_1 : N \rightarrow S \) is a function that maps a node to a state,

6. \( \mu_2 : E \rightarrow T \) is a function that maps an edge to a transition,\(^1\)

7. \( \mu_3 \) is a function that maps an edge to a timing constraint between nodes,

satisfying that for every execution \((s_0, \vec{x}_0, t_0) \xrightarrow{\tau} (s_1, \vec{x}_1, t_1) \xrightarrow{\tau} (s_2, \vec{x}_2, t_2) \cdots\), there exists a sequence \( e_0 n_1 e_1 n_1 e_2 n_2 \cdots \) such that \( \mu_1(n_i) = s_i, \mu_2(e_i) = \tau_i, \mu_3(e_i) = \text{time-rel}(e_i, e_0 e_1 \cdots e_{i-1}) \) for all \( i \).

For a node \( n_i \), we say that \( n_i \) is reachable through sequence \( e_0 e_1 \cdots e_{i-1} e_i \).

**Lemma 4.1** For a sequence \( e_0 e_1 e_2 \cdots e_k \) of edges in \( G \), the condition

\[
(\bigwedge_{1 \leq j \leq k} \mu_3(e_j)) \land (\bigwedge_{1 \leq j \leq k} (\forall (e_{j-1}) \leq \forall (e_j)))
\]

\[
\equiv (\bigwedge_{1 \leq j \leq k} \text{time-rel}(e_j, e_0 \cdots e_{j-1})) \land (\bigwedge_{1 \leq j \leq k} (\forall (e_{j-1}) \leq \forall (e_j)))
\]

\[
\equiv (\bigwedge_{1 \leq j \leq k} (\bigwedge_{1 \leq l \leq l_{j}} (\forall (e_j) \land \forall (e_{x_{ji}} + c_{ji})))) \land (\bigwedge_{1 \leq j \leq k} (\forall (e_{j-1}) \leq \forall (e_j)))
\]

is satisfiable, where \( \pi_1(\mu_3(e_j)) = (\land_{1 \leq i \leq l_j} (x_{ji} \land c_{ji})) \).

**Proof.** The proof follows from Definition 3.2 and Definition 4.1. \(\square\)

The satisfiability of the above condition means that the target node of \( e_k, \text{target}(e_k) \) is reachable through sequence \( e_0 e_1 \cdots e_k \). The first part of the condition says that enabling conditions are satisfied and the second part indicates that the time ordering is preserved.

The following lemma insures that behaviors of the timed automaton can be obtained from the corresponding reachability tree.

**Lemma 4.2** For a timed automaton \( M \) and the corresponding reachability tree \( G \), suppose there is a sequence \( e_0 e_1 e_2 \cdots \) from the root in \( G \). For every timed sequence \( t_0 t_1 t_2 \cdots \) such that \( t_0 = 0 \) and \( t_i \leq t_{i+1} \), if \( \mu_3(e_i)[\forall (e_0)/t_0, \forall (e_1)/t_1, \forall (e_2)/t_2, \ldots] \) is true for all \( i \), then the sequence

\[
((\pi_3(\mu_2(e_1)), t_1), (\pi_3(\mu_2(e_2)), t_2), \ldots)
\]

is a behavior in \( M \).

**Proof.** This follows immediately from Definition 3.3, Definition 4.1 and Lemma 4.1. \(\square\)

For a sequence \( e_0 e_1 e_3 e_5 \) of the reachability tree in Figure 2(a), \( \mu_3(e_1) \land \mu_3(e_3) \land \mu_3(e_5) \) is equal to condition \( \forall (e_1) \geq \forall (e_0) + 4 \land \text{true} \land \forall (e_5) \leq \forall (e_0) + 6 \). Since the condition is satisfiable (e.g. \( \forall (e_0):=0, \forall (e_1):=45, \forall (e_3):=5, \) and \( \forall (e_5):=6 \)), node \( n_5 \) is reachable and \((a, 4.5), (b, 5), (c, 6)\) is a behavior of the timed automaton.

\(^1\)This component has redundant information with \( E \). But, it is added for convenience.
4.2 The Timed Reachability Graph

Suppose a timed automaton $M$ and its corresponding reachability tree $G$. Since $G$ may have infinitely many nodes and edges, we need to construct its finite representation, a timed reachability graph. Our approach of generating the timed reachability graph is to cluster a (possibly infinite) set of nodes that have the same reachability property.

**Definition 4.2** For a reachability tree $G = (N, n_{\text{init}}, E, \mu_1, \mu_2, \mu_3)$, a binary relation $R \subseteq N \times N$ is an $r$-equivalence relation if, for every $(n_1, n_2) \in R$,
\begin{enumerate}
    \item two nodes are labeled with the same state ($\mu_1(n_1) = \mu_1(n_2)$) and
    \item for every edge $e_1$ with source$(e_1) = n_1$, there exists $e_2$ such that source$(e_2) = n_2$, $\mu_2(e_1) = \mu_2(e_2)$ and (target$(e_1)$, target$(e_2)$) $\in R$, and vice versa.
\end{enumerate}

If two nodes $n_1$ and $n_2$ are in some $r$-equivalence relation $R$, $n_1$ and $n_2$ are said to be equivalent with respect to reachability, denoted by $n_1 \sim R n_2$.

For a reachability tree $G$, we define some notations as follows. For two edges $e_1$ and $e_2$ with $\@_1(e_1) \leq \@_1(e_2)$, let $\text{distance}(e_1, e_2)$ represent the earliest time of the execution of $e_2$ after $e_1$ was executed and $\text{distance}(e_2, e_1)$ represent the earliest time of the execution of $e_1$ before the execution of $e_2$. For a clock $x$ and an edge $e_2$, $\text{distance}(x, e_2)$ is defined as the earliest time of the execution of $e_2$ after $x$ was reset. Note that if $e_1$ is the last edge in $\text{Reach}(e_2)$ that reset $x$, then $\text{distance}(x, e_2)$ is equal to $\text{distance}(e_1, e_2)$. For a clock $x$ and a set $T_s \subseteq T$ of transitions, let $c(x, T_s)$ be the largest constant that is compared with $x$ in all of the enabling conditions of the transitions in $T_s$.

**Definition 4.3** For two nodes $n$ and $n'$ in $G$ such that $\mu_1(n) = \mu_1(n')$, let $e$ and $e'$ be the incoming edges of $n$ and $n'$, respectively. For a set $T_s \subseteq T$, let $X_s = \{x_1, \ldots, x_k\}$ be the set of clocks appearing in the enabling conditions in $T_s$. Let $e_{x_i}$ be the last edge in $\text{Reach}(e)$ that reset $x_i$ and $e'_{x_i}$ be the last edge in $\text{Reach}(e')$ that reset $x_i$. Let $d = \max\{c(x, T_s) | x_i \in X_s\}$. The the $r$-equivalence condition for $n$ and $n'$ over $T_s$ is defined as follows:
\begin{enumerate}
    \item for all $i$, either
        \[ \text{distance}(x_i, e) = \text{distance}(x_i, e') \] or
        \[ \text{distance}(x_i, e) > c(x_i, T_s) \text{ and } \text{distance}(x_i, e') > c(x_i, T_s), \] and
    \item for all $i$ and $j$ such that $\text{distance}(x_i, e) \leq c(x_i, T_s)$ and $\text{distance}(x_j, e) \leq c(x_j, T_s)$, either
        \[ \text{distance}(e_{x_i}, e_{x_j}) = \text{distance}(e'_{x_i}, e'_{x_j}) \] or
        \[ \text{distance}(e_{x_i}, e_{x_j}) < -d \text{ and } \text{distance}(e'_{x_i}, e'_{x_j}) < -d. \]
\end{enumerate}

**Lemma 4.3** For two nodes $n$ and $n'$ in $G$ such that $\mu_1(n) = \mu_1(n')$, if the $r$-equivalence condition for $n$ and $n'$ over $T$ holds, then $n_1 \sim R n_2$. 


Proof. Let us construct a set $R \subseteq N \times N$. Initially, $R = \{(n, n')\}$. For $(n_1, n'_1) \in R$ and for every $e_1, e'_1$ such that $\text{source}(e_1) = n_1$, $\text{source}(e'_1) = n'_1$ and $\mu_2(e_1) = \mu_2(e'_1)$, $R = R \cup \{(\text{target}(e_1), \text{target}(e'_1))\}$. We now show that $R$ is an $r$-equivalence relation. For all $(n_1, n'_1) \in R$, the $r$-equivalence condition holds since the condition holds for $(n, n')$ and its descendents preserve the condition by the $r$-equivalence condition 2 (with $T_s = T$). For every $e_1$ such that $\text{source}(e_1) = n$, there exists $e'_1$ such that $\text{source}(e'_1) = n'$ and $\mu_2(e_1) = \mu_2(e'_1)$ by the $r$-equivalence condition 1, and vice versa. Thus, $R$ is an $r$-equivalence relation, that is, $n_1 \sim_r n_2$. □

We can cluster equivalent nodes using Lemma 4.3. Note that if we can find a $r$-equivalence that equates more nodes, then the resulting reachability graph becomes much smaller. To find such an $r$-equivalence, we use the following observation: Since we only need to consider transitions whose enabling conditions depend on the values of clocks in a node $n$, it may be possible to use $T_s$ (in Definition 4.3) that is a proper subset of $T$. For a node $n$ and a transition $\tau$, the values of clocks in $n$ do not affect $\tau$ if $\tau$ is in one of the following cases:

1. $\tau$ is unreachable from state $\mu_1(n)$, that is, there is no path from state $\mu_1(n)$ to a state whose outgoing transitions include $\tau$.

2. For every path from state $\mu_1(n)$ to a state whose outgoing transitions include $\tau$, each clock used in the enable condition of $\tau$ is reset to zero in some transition along the path.

Based on the above facts, we provide weaker conditions to equate two nodes as follows. For a node $n$, let $RT(n)$ denote the set of transitions in $T$ which are not in the above cases. That is, $RT(n)$ includes only transitions, say $\tau$, such that $\tau$ is reachable from $\mu_1(n)$ and there is a path from $\mu_1(n)$ to $\tau$ such that there is a clock used in $\tau$ that is not reset in the path.

Lemma 4.4 For two nodes $n$ and $n'$ in $G$ such that $\mu_1(n) = \mu_1(n')$, if the $r$-equivalence condition for $n$ and $n'$ over $RT(n)$ holds, then $n_1 \sim_r n_2$.

Proof. This can be proved similarly to Lemma 4.3. □

In a reachability graph, timing constraints are expressed in the form of $\langle (e) \ast \langle \{e_1, ..., e_l\} + c$ rather than $\langle (e) \ast \langle (e') + c$ since multiple paths reaching to $e$ can exist due to cycles. A timing constraint $\langle (e) \leq \langle \{e_1, e_2\} + 5$ means that the time when the system executes $e$ is less than or equal to the last time when the system executed $e_1$ or $e_2$.

We now present an algorithm that constructs a reachability graph of a given timed automaton. The algorithm is given in Figure 3. In the algorithm, Step 1 is an initialization. $N$ includes the set of nodes and $E$ includes the set of edges in $G$. Unexplored is the set of nodes which need to be tested whether there exists its equivalent node in $N$. It is implemented using a queue in order to explore the reachability graph by breadth first search. BackEdge is the set of back-edges which make cycles in the graph. Step 2 repeats as long as there is at least one unexplored reachable node. In Step 2A, if there exists a node $n'$ satisfying conditions of Lemma 4.4, then the selected node $n$ is removed and the outgoing edge is adjusted to the equivalent node $n'$. In Step 2B, if there is no such node, then the selected node $n$ is added
1. 
\[ N := \emptyset; \ E := \emptyset; \]
create the initial node \( n_{\text{init}} \) such that \( \mu_1(n_{\text{init}}) = s_{\text{init}}; \)
\( Unexplored := \{n_{\text{init}}\}; \)

2. 
\textbf{while} \( Unexplored \neq \emptyset \) \textbf{do}
\begin{enumerate}
  \item \textbf{pick and remove a node} \( n \) \textbf{from} \( Unexplored; \)
  \item[A] \textbf{if} there exists a node \( n' \) in \( N \) such that \( n \sim \tau n' \) \textbf{then}
  \begin{itemize}
    \item remove the incoming edge \( e \) of \( n \) from \( E; \)
    \item create an edge \( e' \) with \( \text{source}(e') := \text{source}(e), \text{target}(e') := n', \)
    \item \( \mu_2(e') := \mu_2(e), \mu_3(e') := \mu_3(e)[e/e']; \)
    \item add \( e' \) into \( E \) and \( \text{BackEdge}; \)
  \end{itemize}
  \item[B] \textbf{else}
  \begin{itemize}
    \item add \( n \) into \( N; \)
    \item for every outgoing transition \( \tau \in T \) of the corresponding state of \( n, \mu_1(n) \) \textbf{do}
      \begin{itemize}
        \item create a node \( n' \) with \( \mu_1(n') := \text{target}((\tau)); \)
        \item if \( n' \) is reachable \textbf{then}
          \begin{itemize}
            \item add \( n' \) into \( Unexplored; \)
            \item create an edge \( e' \) with \( \text{source}(e') := n, \text{target}(e') := n', \mu_2(e') := \tau, \)
            \item \( \mu_3(e') := \text{time.rel}(\pi_1(\tau), e', \text{Reach}(e)). \)
            \item add \( e' \) into \( E; \)
          \end{itemize}
        \end{itemize}
      \end{itemize}
    \end{itemize}
  \end{enumerate}
\textbf{end while}

3. 
\textbf{for} every edge \( e \) reachable from \( \text{BackEdge} \) \textbf{do}
\begin{itemize}
  \item update the timing relation \( \mu_3(e) \) appropriately;
\end{itemize}
\textbf{end for}

Figure 3: Reachability Graph Construction Algorithm
to N. Nodes corresponding to states immediately reachable from \( pl(n) \) are created and added to Unexplored. After the while loop, Step 3 adjusts timing constraints in edges reachable by BackEdge. For each \( e \in \text{BackEdge} \) and each \( e' \in E \), if \( e' \) is reachable from \( e \) and its timing constraint \( \@{e'} \star \@{\ldots} + c \) is affected by \( e'' \in \text{Reach}(e) \), then \( e'' \) is added to the constraint like \( \@{e'} \star \@{\ldots, e'', \ldots} + c \).

**Lemma 4.5** The reachability graph generated from the above algorithm is finite.

**Proof.** Considering Lemma 4.3, \( \text{distance}(x_i, e) \) is represented as at most \( (c(x_i) + 2) \) different values and \( \text{distance}(x_i, x_j) \) is represented as at most \( (d + 2)^2 \) different values. Since the reachability graph generated using Lemma 4.4 is smaller than one generated using Lemma 4.3, the algorithm generates the finite reachability graph. □

We illustrate the algorithm using an example of the timed automaton in Figure 1. The timed reachability graph generated from the algorithm is shown in Figure 2(b). Initially, \( N := \emptyset \), \( E := \{e_0\} \), Unexplored = \{no\}, and BackEdge := \emptyset. Each step represents the execution of the while-loop body in Step 2.

1. Select \( no \) from Unexplored. After executing Step 2A, \( N := \{no\}, \) Unexplored := \{n1, n2\}, \( E := \{e_0, e_1, e_2\} \).

2. For \( n_1, n_2 \in \text{Unexplored} \), process Step 2. Then \( N := \{n_0, n_1, n_2\}, \) Unexplored := \{n3\}, \( E := \{e_0, e_1, e_2, e_3\} \).

3. For \( n_3 \), check whether it is equivalent to \( n_0 \). \( RT(s_0) = \{r_1, r_2, r_3\}, RX(s_0) = \{x, y\} \), and \( c(y, RT(s_0)) = 6 \). Since \( \text{distance}(y, e_0) = 0 \) and \( \text{distance}(y, e_3) = 4 \), \( n_0 \sim \not{\sim} n_3 \). Thus, \( N := \{n_0, n_1, n_2, n_3\}, \) Unexplored := \{n4, n5\} and \( E := \{e_0, e_1, e_2, e_3, e_4, e_5\} \).

4. Considering \( n_4 \), \( RT(s_1) = \{r_2\}, RX(s_1) = \{y\} \), and \( c(y, RT(s_1)) = 6 \). \( n_4 \) is not equivalent to \( n_1 \) since \( \text{distance}(y, e_1) = 4 \) and \( \text{distance}(y, e_7) = 8 \). At the last, \( N := \{n_0, n_1, n_2, n_3, n_4\}, \) Unexplored := \{n5, n6\}, \( E := \{e_0, e_1, e_2, e_3, e_4, e_5, e_6\} \).

5. Considering \( n_5, n_2 \sim \not{\sim} n_5 \). Thus, a new edge \( e'_5 \) such that \( \mu_2(e'_5) = \mu_2(e_5), \mu_3(e'_5) = \mu_3(e_5)[e_5/e'_5] \) are created. \( N := \{n_0, n_1, n_2, n_3, n_4\}, \) Unexplored := \{n6\}, \( E := \{e_0, e_1, e_2, e_3, e_4, e'_5, e_6\} \). And BackEdge becomes \{e'_5\}.

6. For \( n_6 \), check whether it is equivalent to \( n_0 \) or \( n_3 \). Since \( \text{distance}(y, e_6) = 8 \), \( n_0 \not{\sim} n_3 \not{\sim} n_6 \) (see the step for \( n_3 \)). \( N := \{n_0, n_1, n_2, n_3, n_4, n_6\}, \) Unexplored := \{n7\}, \( E := \{e_0, e_1, e_2, e_3, e_4, e'_5, e_6, e_7\} \).

7. For \( n_7 \), since \( \text{distance}(y, e_4) = 8 \) and \( \text{distance}(y, e_7) = 12 \) are all greater than \( c(y, RT(s_1)) \) and \( \text{distance}(x, e_4) = \text{distance}(x, e_7) = 0 \), \( n_4 \sim \not{\sim} n_7 \). Thus, we can remove the edge from \( n_6 \) to \( n_7 \) and put a back-edge \( e'_7 \) from \( n_6 \) to \( n_4 \) with \( \mu_2(e'_7) := \mu_2(e_7) \) and \( \mu_3(e'_7) := \mu_3(e_7)[e_7/e'_7] \). That is, \( N := \{n_0, n_1, n_2, n_3, n_4, n_6\}, \) Unexplored := \emptyset, \( E := \{e_0, e_1, e_2, e_3, e_4, e_5, e_6, e'_7\}, \) BackEdge := \{e'_5, e'_7\}. |
In Step 3, there exist back-edges. Since \( x \) is reset on \( e'_f \), the execution time of \( e'_f \) may affect the execution time of next reachable edges, \( e_6 \) and \( e'_f \). The timing constraint of \( \mu_2(e'_f) \) is \( x \geq 4 \). Thus we augment the timing of \( e'_f \) with \( \oplus(e'_f) \geq \oplus\{e_4,e'_f\} + 4 \).

**Implementation Issue.** A sequence \( e_1, e_2, ..., e_l \) is said to be **closed** if edges appearing in the timing constraints from \( e_2 \) to \( e_1 \) are included in the set of edges in the sequence. For a closed sequence of edges, \( e_1, e_2, ..., e_l \), the corresponding weighted graph \( W \) is defined as follows. The graph \( W \) includes nodes \( m_i \) corresponding to \( e_i \) for all \( 1 \leq i \leq l \) and weighted edges from \( w_i \) to \( w_{i+1} \) with weight 0 for all \( 1 \leq i < l \). The following weighted edges are also included in \( W \): For each relation \( \oplus(e_i) \ast \{\oplus(e_{i_1}), \oplus(e_{i_2}),...,\oplus(e_{i_m})\} + c \) in the timing constraint \( \mu_3(e_i) \), suppose \( e_j \) is the nearest edge among \( e_{i_1}, e_{i_2}, ..., e_{i_m} \) backward from \( e_i \) in the given sequence.

- If \( \ast = \leq \) then put an edge from \( w_i \) to \( w_j \) with weight \( c \).
- If \( \ast = \geq \) then put an edge from \( w_j \) to \( w_i \) with weight \( -c \).

The value of \( \text{distance}(x, e) \) in Lemma 4.4 can be obtained by computing the maximum distance from an edge with resetting clock \( x \) to \( e \) in the weighted graph of the path up to \( e \). The reachability test (the satisfiability of the condition in Lemma 4.1) of a node \( n \) through a path \( seq = e_0, ..., e_l \) from the root can be done using the weighed graph \( W \) of \( seq \) as well as symbolic computation. If the maximum weight \( \text{distance}(e_0, e_l) \) from \( e_0 \) and \( e_l \) is greater than or equal to zero, then \( n \) is reachable.

### 5 Minimum and Maximum Time Bounds Between Two Actions

An advantage of the reachability graph is that algorithms can be developed for proving properties of the graph and this implies proving properties of the original system. In this section, we give a procedure to compute the earliest time and the latest time that an action can succeed another one with respect to the reachability graph.

For two actions \( a \) and \( b \), let \( \text{min}(a, b) \) and \( \text{max}(a, b) \) denote the earliest time and the latest time at which \( b \) can happen after \( a \), respectively. For a sequence \( p \), \( \text{first}(p) \) denotes the first element of \( p \) and \( \text{last}(p) \) denotes the last element of \( p \). We give the algorithm shown in Figure 4 for computing \( \text{min}(a, b) \) and \( \text{max}(a, b) \) with respect to a given reachability graph. Step 1 finds all non-cyclic sequences \( p = e_1e_2...e_k \) such that \( a \) is performed on \( e_1 \), and \( b \) is not performed during the intermediate steps and is performed on \( e_k \). Step 2 finds the set \( Q \) of the least closed sequences containing \( p \). The sequences can be computed by back-tracing the graph from \( e_1 \). It may happen to infinitely back-tracing the graph from \( e_1 \) to find the least closed sequence including \( p \) due to cycles in the graph. Thus, Step 2 may not terminate. We are currently investigating the termination of the procedure. For every \( q \in Q \), the maximum weight from \( e_1 \) to \( e_k \), \( d(e_1, e_k) \), is equal to the minimum time bound from \( a \) and \( b \) in \( q \), and the absolute number
1. 
\[ P := \{ (e_1 e_2 \ldots e_k) \mid \text{non-cyclic, } \pi_3(\mu_2(e_1)) = a, \pi_3(\mu_2(e_k)) = b, \forall 1 < i < k. \pi_3(\mu_2(e_i)) \neq b \}; \]

2. 
\[ Q := \emptyset; \]
\[ \text{for all } p \in P \text{ do} \]
\[ Q_1 := \{ q \mid q \text{ is the least closed sequence including } p \}; \]
\[ \text{for all } q \in Q_1 \text{ do} \]
\[ f(q) := \text{first}(p); \]
\[ l(q) := \text{last}(p); \]
\[ \text{end for} \]
\[ Q := Q \cup Q_1; \]
\[ \text{end for} \]

3. 
\[ \text{for all } q \in Q \text{ do} \]
\[ \text{construct the weighted graph corresponding to } q; \]
\[ \text{add } d(f(q), l(q)) \text{ into } F_{\text{min}}; \]
\[ \text{add } |d(l(q), f(q))| \text{ into } F_{\text{max}}; \]
\[ \text{end for} \]

4. 
\[ \min(a, b) := \text{minimum}\{F_{\text{min}}\}; \]
\[ \max(a, b) := \text{maximum}\{F_{\text{max}}\}; \]

Figure 4: Procedure for Computing \( \min(a, b) \) and \( \max(a, b) \)

of the maximum weight from \( e_k \) to \( e_1 \), \( d(e_k, e_1) \), is equal to the maximum time bound from \( a \) to \( b \). In Step 3, \( d(e_1, e_k) \) and \( d(e_k, e_1) \) can be computed using the weighted graph corresponding to \( q \). After Step 3, \( \min(a, b) \) is the minimum of \( F_{\text{min}} \) and \( \max(a, b) \) is the maximum of \( F_{\text{max}} \).

The minimum and maximum time bounds between two given states can be computed by slightly modifying the algorithm in Figure 4.

6 An Example: Railroad Crossing

The standard railroad crossing problem has been used to compare different formal methods for real-time systems [7]. Figure 5 shows an automatic controller that opens and closes a gate at a railroad crossing presented in [3]. The system is formed as the composition of three components which execute in parallel and synchronize with the same action names. When a train approaches the crossing, it sends an approach signal to the controller and enters the crossing at least 200 seconds later. When a train leaves the crossing, it sends an exit signal to the controller. The exit signal should be sent within 500 seconds after the approach signal. The controller sends a signal lower to the gate exactly at 100 seconds after the approach signal and sends a raise signal within 100 seconds after exit. The gate responds to lower and raise signals by moving
down within 100 seconds and moving up between 100 and 200 seconds, respectively. These three components can be composed into a global timed automaton. From the automaton, we can compute the reachability graph as shown in Figure 6, where we compute a utility property: whenever the gate goes down, what the earliest time and the latest time at which it is moved back up are, that is, \( \min(down, up) \) and \( \max(down, up) \). They are computed as follows using the algorithm in Figure 4.

**Step1** There are two non-cyclic sequences with the first action down and the last action up:

\[
p_1 = e_4 e_7 e_9 e_{11} e_{12} e_{16} \quad \text{and} \quad p_2 = e_4 e_7 e_9 e_{11} e_{12} e_{13} e_{17}.
\]

**Step2** The least closed sequences containing \( p_1 \) and \( p_2 \) are

\[
\begin{align*}
p_{11} &= e_1 e_2 e_4 e_7 e_9 e_{11} e_{12} e_{16}, & p_{12} &= e_{12} e_{13} e_{17} e_2 e_4 e_7 e_9 e_{11} e_{12} e_{16}, \\
p_{21} &= e_1 e_2 e_4 e_7 e_9 e_{11} e_{12} e_{13} e_{17}, & p_{22} &= e_{12} e_{13} e_{17} e_2 e_4 e_7 e_9 e_{11} e_{12} e_{13} e_{17}.
\end{align*}
\]

**Step3** For the sequence \( p_{11} \), Figure 7 shows the weighted graph, where \( \min(e_4, e_{16}) = 100 \) and \( \max(e_4, e_{16}) = 700 \). Similarly for \( p_{12}, p_{21}, \) and \( p_{22} \).

**Step4** \( \min(down, up) = 100 \) and \( \max(down, up) = 700 \).

Courcoubetis and Yannakakis [6] give an algorithm to compute the minimum and the maximum time bounds between two states with respect to a region graph whose complexity is propositional to the size of the region graph. For the railroad crossing example, the number of regions is greater than \( 10^7 \). Moreover, the algorithm gives the results as the function of \( \epsilon \ll 1 \) since the precise time information is lost in a region graph. In [3], they verify a property:
Figure 6: A Reachability Graph

Figure 7: Weighted Graphs
whenever the gate goes down, it is moved back up within $K$ for some $K$. When $K = 500$, they construct a minimal reachable region graph whose number of regions is 412. Moreover, a different minimal graph is required to be computed for a different value of $K$ in their approach. However, our approach generates the timed reachability graph with 15 nodes as shown in Figure 6. With $\text{min}(\text{down, up}) = 100$ and $\text{max}(\text{down, up}) = 700$, we can say that for $100 \leq K \leq 700$, whenever the gate goes down, it is moved back up within $K$.

7 Conclusion

We have presented an algorithm to cope with the state explosion problem in generating the state space of a timed automaton. Our algorithm clusters a set of states that are equivalent under the notion of r-equivalence, which we have formally defined. In our experience, the timed reachability graph is much smaller than the minimal reachable region graph of [4]. To show the usefulness of the timed reachability graph, we have presented a procedure for computing the minimum and maximum time bounds between two actions. As an illustration, we have applied our technique to the well-known railroad crossing example.

Although the timed reachability graph presented in this paper is similar to the computation graph in [8], there are several differences. The underlying time domain of the computation graph is discrete. In the computation graph, each node represents a transition not a state. While timing information in a computation graph is maintained separately using a weighted graph called a separation graph, we represent timing information using relative time relations between edges in the timed reachability graph. The relations make it possible to test node's reachability using a symbolic operation in Lemma 4.1. Since a Modechart specification has no clock, they have no equivalence condition comparing distances between clocks and thus have an simpler equivalence condition. In particular, timing information is defined only between two nodes that have an edge between them. This limitation is natural since in a Modechart specification, timing constraints can be expressed between two sequential modes with a transition.

The work described in this paper is part of our research in developing effective tools based on state space exploration [5]. So far, we have developed Communicating Timed State Machines (CTSM) as a formalism that can be used to design and implement analysis algorithms based on state-space exploration. CTSM is a state machine including one-to-many communication, message passing, data variables, and real-time. We have developed a minimization procedure with respect to bisimulation for states with arbitrary data variables [9]. For the specification of timing constraints in CTSM, we follow the strategy of timed automata and can use the timed reachability analysis procedure described in this paper. We plan to integrate two procedures to construct the state generation of CTSM. We are also currently investigating other properties that can be checked directly from the reachability graph generated by our algorithm.
References


