Timed Automata

Rajeev Alur
University of Pennsylvania and Bell Labs
alur@cis.upenn.edu

Abstract

Model checking is emerging as a practical tool for automated debugging of complex reactive systems such as embedded controllers and network protocols (see [CK96] for a survey). In model checking, a high-level description of a system is compared against a logical correctness requirement to discover inconsistencies. Traditional techniques for model checking do not admit an explicit modeling of time, and are thus, unsuitable for analysis of real-time systems whose correctness depends on relative magnitudes of different delays. Consequently, timed automata [AD94] were introduced as a formal notation to model the behavior of real-time systems. Its definition provides a simple, and yet general, way to annotate state-transition graphs with timing constraints using finitely many real-valued *clock variables*. Automated analysis of timed automata relies on the construction of a finite quotient of the infinite space of clock valuations. Over the years, the formalism has been extensively studied leading to many results establishing connections to circuits and logic, and much progress has been made in developing verification algorithms, heuristics, and tools. This paper provides a survey of theory of timed automata, and their role in specification and verification of real-time systems.

1 Modeling

Transition systems

We model systems by state-transition graphs whose transitions are labeled with event symbols. A *transition system* S is a tuple $\langle Q, Q^0, \Sigma, \rightarrow \rangle$, where Q is a set of states, $Q^0 \subseteq Q$ is a set of initial states, Σ is a set of labels (or events), and $\rightarrow \subseteq Q \times \Sigma \times Q$ is a set of transitions. For a transition $\langle q, a, q' \rangle$ in $\rightarrow$, we write $q \xrightarrow{a} q'$. The system starts in an initial state, and if $q \xrightarrow{a} q'$ then the system can change its state from q to q' on event a. We write $q \rightarrow q'$ if $q \xrightarrow{a} q'$ for some label a. The state q' is reachable from the state q if $q \rightarrow^* q'$. The state q is a reachable state of the transition system if q is reachable from some initial state.

A complex system can be described as a product of interacting transition systems. Let $S_1 = \langle Q_1, Q_1^0, \Sigma_1, \rightarrow_1 \rangle$ and $S_2 = \langle Q_2, Q_2^0, \Sigma_2, \rightarrow_2 \rangle$ be two transition systems. Then, in the product of S_1 and S_2, a state is a pair (q, q') with $q \in Q_1$ and $q' \in Q_2$. The transitions of the product are labeled with symbols in $\Sigma_1 \cup \Sigma_2$. For a label a, to obtain a-labeled transitions of the product, we require each component system with a in its label-set to execute an a-labeled transition. Formally, the *product*, denoted $S_1 \| S_2$, is the transition system $\langle Q_1 \times Q_2, Q_1^0 \times Q_2^0, \Sigma_1 \cup \Sigma_2, \rightarrow \rangle$ where $(q_1, q_2) \xrightarrow{a} (q_1', q_2')$ iff either (i) $a \in \Sigma_1 \cap \Sigma_2$ and $q_1 \xrightarrow{a}_1 q_1'$ and $q_2 \xrightarrow{a}_2 q_2'$, or (ii) $a \in \Sigma_1 \setminus \Sigma_2$ and $q_1 \xrightarrow{a}_1 q_1'$ and $q_2' = q_2$, or (iii) $a \in \Sigma_2 \setminus \Sigma_1$ and $q_2 \xrightarrow{a}_2 q_2'$ and $q_1' = q_1$. Observe that the symbols that belong to the alphabets of both the automata are used for synchronization. In this definition, synchronization is

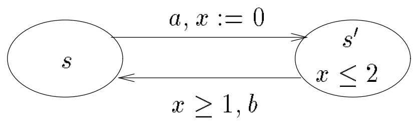


Figure 1: A sample timed automaton

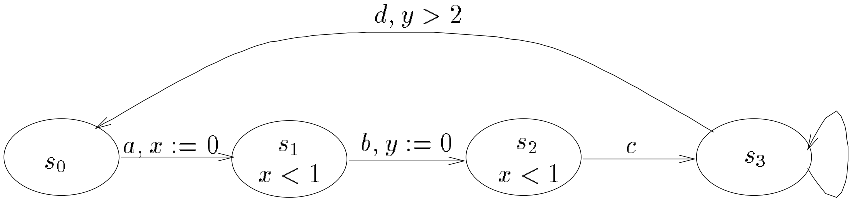


Figure 2: A timed automaton with 2 clocks

blocking, for a common symbol a, a component can execute an a-labeled switch only if the other component can also do so.

Transition systems with timing constraints

To express system behaviors with timing constraints, we consider finite graphs augmented with a finite set of (real-valued) *clocks*. The vertices of the graph are called *locations*, and edges are called *switches*. While switches are instantaneous, time can elapse in a location. A clock can be reset to zero simultaneously with any switch. At any instant, the reading of a clock equals the time elapsed since the last time it was reset. With each switch we associate a clock constraint, and require that the switch may be taken only if the current values of the clocks satisfy this constraint. With each location we associate a clock constraint called its *invariant*, and require that time can elapse in a location only as long as its invariant stays true. Before we define the timed automata formally, let us consider some examples.

Consider the timed automaton of Figure 1. The initial location is s. There is a single clock x. The initial location has no invariant constraint which means that the system can spend arbitrary amount of time in location s. When the system switches to location s' on symbol a, the clock x gets reset to 0. While in location s', the value of the clock x shows the time elapsed since the occurrence of the last switch. The switch from location s' to s is enabled only if this value is greater than 1. The invariant $x \leq 2$ associated with the location s' specifies the requirement that the system can stay in location s' for at most 2 units, and a switch must occur before the invariant is violated. Thus the timing constraint expressed by this automaton is that the delay between a and the following b is always between 1 and 2.

Having multiple clocks allows multiple concurrent delays, as illustrated in Figure 2. The clock x gets set to 0 each time the system switches from s_0 to s_1 on symbol a. The invariant $(x < 1)$ associated with the locations s_1 and s_2 ensures that c-labeled swithc from s_2 to s_3 happens within time 1 of the preceding a. Resetting another independent clock y together with the b-labeled switch from s_1 to s_2 and checking its value on the d-labeled switch from s_3 to s_0 ensures that the delay between b and the following d is always greater than 2. Since the location s_3 has no invariant constraint, the event d can pe postponed indefinitely, and is not guaranteed to happen.

Notice that in the above example, to constrain the delay between a and c and between b and d the system does not put any explicit bounds on the time difference between a and the following b, or c and the following d. This is an important advantage of having multiple clocks which can be set independently of one another. We remark that the clocks of the system do not necessarily correspond to the local clocks of different components in a distributed system. They are fictitious clocks invented to express the timing properties of the system, and all the clocks increase at the uniform rate counting time with respect to a fixed global time frame. Alternatively, we can consider the system to be equipped with a finite number of stop-watches which can be started and checked independently of one another, but all stop-watches refer to the same clock.

Clock constraints and clock interpretations

To define timed automata formally, we need to say what type of clock constraints are allowed as invariants and enabling conditions. An atomic constraint compares a clock value with a time constant, and a clock constraint is a conjunction of atomic constraints. Any value from $\mathbb{Q}$, the set of nonnegative rationals, can be used as a time constant. Formally, for a set X of clock variables, the set $\Phi(X)$ of *clock constraints* φ is defined by the grammar

$$\varphi := x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2,$$

where x is a clock in X and c is a constant in $\mathbb{Q}$.

A *clock interpretation* ν for a set X of clocks assigns a real value to each clock; that is, it is a mapping from X to the set $\mathbb{R}$ of nonnegative reals We say that a clock interpretation ν for X satisfies a clock constraint φ over X iff φ evaluates to true according to the values given by ν. For $\delta \in \mathbb{R}$, $\nu + \delta$ denotes the clock interpretation which maps every clock x to the value $\nu(x) + \delta$. For $Y \subseteq X$, $\nu[Y := 0]$ denotes the clock interpretation for X which assigns 0 to each $x \in Y$, and agrees with ν over the rest of the clocks.

Syntax and semantics

A *timed automaton* A is a tuple $\langle L, L^0, \Sigma, X, I, E \rangle$, where

- L is a finite set of locations,
- $L^0 \subseteq L$ is a set of initial locations,
- Σ is a finite set of labels,
- X is a finite set of clocks,
- I is a mapping that labels each location s in L with some clock constraint in $\Phi(X)$, and
- $E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L$ is the set of switches. A switch $\langle s, a, \varphi, \lambda, s' \rangle$ represents a transition from location s to location s' on symbol a. φ is a clock constraint over X that specifies when the switch is enabled, and the set $\lambda \subseteq X$ gives the clocks to be reset with this switch,

The semantics of a timed automaton A is defined by associating a transition system S_A with it. A state of S_A is a pair (s, ν) such that s is a location of A and ν is a clock interpretation for X such that ν satisfies the invariant $I(s)$. The set of all states of A is denoted Q_A. A state (s, ν) is an initial state if s is an initial location of A and $\nu(x) = 0$ for all clocks x. There are two types of transitions in S_A:

- State can chage due to elapse of time: for a state (s, ν) and a real-valued time increment $\delta \geq 0$, $(s, \nu) \xrightarrow{\delta} (s, \nu + \delta)$ if for all $0 \leq \delta' \leq \delta$, $\nu + \delta'$ satisfies the invariant $I(s)$.
- State can change due to a location-switch: for a state (s, ν) and a switch $\langle s, a, \varphi, \lambda, s' \rangle$ such that ν satisfies φ, $(s, \nu) \xrightarrow{a} (s', \nu[\lambda := 0])$.

Thus, S_A is a transition system with label-set $\Sigma \cup \mathbb{R}$. For instance, for the timed automaton of Figure 2, the state-space of the associated transition system is $\{s_0, s_1, s_2, s_3\} \times \mathbb{R}^2$, the label-set is $\{a, b, c, d\} \cup \mathbb{R}$, and some sample transitions are

$$(s_0, 0, 0) \xrightarrow{1.2} (s_0, 1.2, 1.2) \xrightarrow{a} (s_1, 0, 1.2) \xrightarrow{0.7} (s_1, 0.7, 1.9) \xrightarrow{b} (s_2, 0.7, 0) \xrightarrow{0.1} (s_2, 0.8, 0.1) \xrightarrow{0.1} (s_2, 0.9, 0.2)$$

Note the time-additivity property of the transition system S_A: if $q \xrightarrow{\delta} q'$ and $q' \xrightarrow{\epsilon} q''$ then $q \xrightarrow{\delta+\epsilon} q''$.

Remark 1 (Multiple events in zero time) By our definition, a transition corresponds to a single event or elapse of time, and arbitrary interleaving of the two types of transitions is allowed. Thus, multiple events can occur without any intervening time-elapse transition. Variants of this scheme are possible (e.g. the original definition in [AD94] requires nozero time to elapse between two location-switches) and do not affect any of the results. ■

Remark 2 (Nonzenoness) In this introductory survey, we have omitted requirements on the definition necessary for executability. First, when the invariant of a location is violated, some outgoing edge must be enabled. Second, from every reachable state, the automaton should admit the possibility of time to diverge. For example, the automaton should not enforce infinitely many events in a finite interval of time. Automata satisfying this operational requirement are called *nonZeno*. The interested reader is referred to [AL91, GSSL94, AH97]. ■

Product construction

We proceed to define a product construction for timed automata so that a complex system can be defined as a product of component systems. Let $A_1 = \langle L_1, L_1^0, \Sigma_1, X_1, I_1, E_1 \rangle$ and $A_2 = \langle L_2, L_2^0, \Sigma_2, X_2, I_2, E_2 \rangle$ be two timed automata. Assume that the clock sets X_1 and X_2 are disjoint. Then, the product, denoted $A_1 \| A_2$, is the timed automaton $\langle L_1 \times L_2, L_1^0 \times L_2^0, \Sigma_1 \cup \Sigma_2, X_1 \cup X_2, I, E \rangle$, where $I(s_1, s_2) = I(s_1) \wedge I(s_2)$ and the switches are defined by:

1. for $a \in \Sigma_1 \cap \Sigma_2$, for every $\langle s_1, a, \varphi_1, \lambda_1, s_1' \rangle$ in E_1 and $\langle s_2, a, \varphi_2, \lambda_2, s_2' \rangle$ in E_2, E contains $\langle (s_1, s_2), a, \varphi_1 \wedge \varphi_2, \lambda_1 \cup \lambda_2, (s_1', s_2') \rangle$.
2. for $a \in \Sigma_1 \backslash \Sigma_2$, for every $\langle s, a, \varphi, \lambda, s' \rangle$ in E_1 and every t in L_2, E contains $\langle (s, t), a, \varphi, \lambda, (s', t) \rangle$.
3. for $a \in \Sigma_2 \backslash \Sigma_1$, for every $\langle s, a, \varphi, \lambda, s' \rangle$ in E_2 and every t in L_1, E contains $\langle (t, s), a, \varphi, \lambda, (t, s') \rangle$.

Thus, locations of the product are pairs of component-locations, and the invariant of a compound location is the conjunction of the invariants of the component locations. The switches are obtained

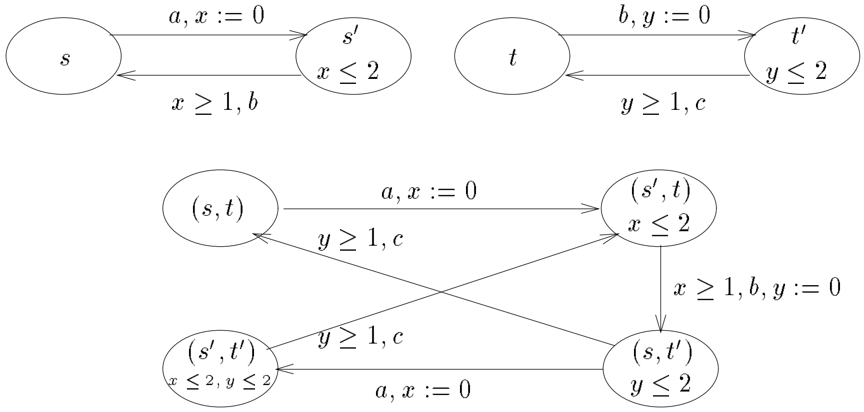


Figure 3: Product construction for timed automata

by synchronizing the switches with identical labels. The product construction is illustrated in Figure 3.

It is easy to verify that transition system associated with the product is the product of the transition systems of the components: $S_{A_1 \| A_2}$ and $S_{A_1} \| S_{A_2}$ are isomorphic. This implies that timed transition systems synchronize not only on switches with common labels, but also on amount of time elapsed.

Remark 3 (Compositionality) For communication between system components, many competing alternatives to blocking synchronization exist. The choice of synchronization primitives is somewhat orthogonal to the problem of analysis of timing constraints. Examples of formalisms that admit modeling real-time systems include timed transition systems [Ost90, HMP94], timed I/O automata [LA92], process algebras such timed CSP [RR88], and Modecharts [JM87]. The algorithmic techniques for timed automata can be applied to these other models also via compilation. To model *open* real-time systems (i.e. those interacting with the environment), one needs to make a distinction between which events are controlled by the system and which events are controlled by the environment. Such a compositional framework provides foundations to decompose the analysis problem into simpler problems [MMT91, AH97]. Issues pertaining to the impact of timing on synchronization are studied in [BST98]. ■

Train-Gate Controller Example

We consider an example of an automatic controller that opens and closes a gate at a railroad crossing. The system is composed of three components: TRAIN, GATE and CONTROLLER as shown in Figure 4. The train communicates with the controller with two events *approach* and *exit*. The events *in* and *out* mark the events of entry and exit of the train from the railroad crossing. The train is required to send the signal *approach* at least 2 minutes before it enters the crossing. This requirement is expressed by the guard $x > 2$ associated with the event *in*. Furthermore, we know that the maximum delay between the signals *approach* and *exit* is 5 minutes. This requirement is expressed by the invariant $x \leq 5$ in the locations s_1, s_2, and s_3. The gate is open in location t_0 and closed in location t_2. It communicates with the controller through the signals *lower* and *raise*. The events *up* and *down* denote the opening and the closing of the gate. The gate responds to the signal *lower* by closing within 1 minute, and responds to the signal *raise* within 1 to 2 minutes.

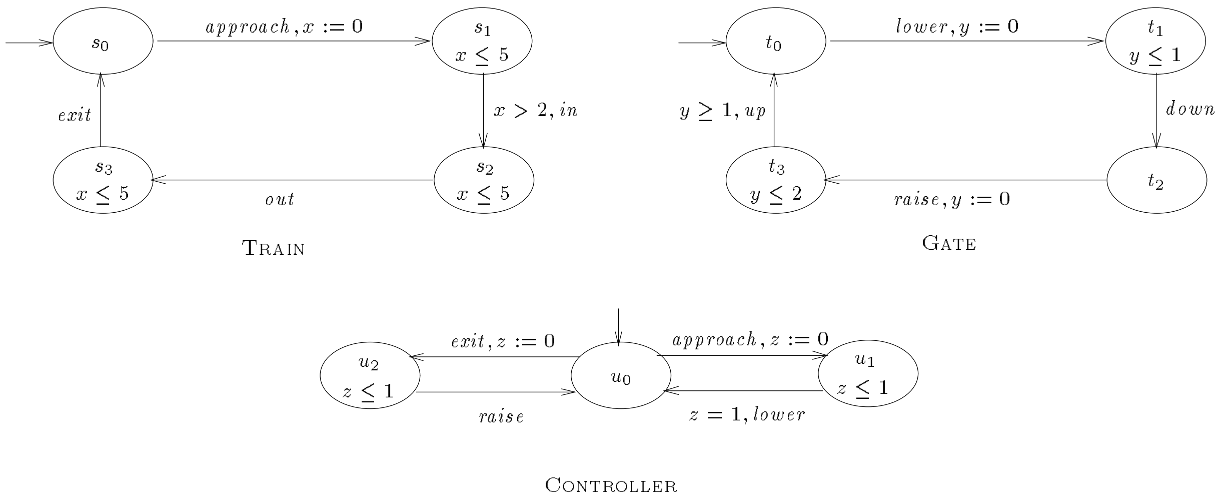


Figure 4: Train-gate controller

The clock y is used to express these constraints. The controller idle location is u_0. Whenever it receives the signal *approach* from the train, it responds by sending the signal *lower* to the gate. Whenever it receives the signal *exit*, it responds with a signal *raise* to the gate. The response time of the controller to the *approach* signal is 1 minute, and to the signal *exit* is at most 1 minute. These constraints are expressed using the clock z. The entire system Grc is then Train || Gate || Controller.

The safety correctness requirement for the system is that whenever the train is inside the gate, the gate should be closed. This corresponds to establishing that in the product system Grc, in every reachable state, if the location of Train is s_2 then the location of Gate should be t_2. Observe that such a location is reachable in the product graph. For example, there is an edge from the initial location (s_0, t_0, u_0) to (s_1, t_0, u_1), and from (s_1, t_0, u_1) to (s_2, t_0, u_1), corresponding to the scenario in which the event *approach* is immediately followed by the event *in*. This is because our product is simply a syntactic operation that annotates product locations with conjunctions of invariants, and product edges with conjunctions of enabling conditions, without any analysis. If we consider the timing information, we can establish that the event *approach* cannot be immediately followed by the event *in*: in the location (s_1, t_0, u_1) both clocks x and z have the same value, and hence the event *lower* with guard $z = 1$ is guaranteed to precede the event *in* with guard $x > 2$. In fact, in the transition system S_{GRC}, no reachable state has Train in location s_2 with Gate not in location t_2. The computational problem in timing verification is to make such deductions by analyzing the timing constraints.

2 Reachability Analysis

A location s of the timed automaton A is said to be reachable if some state q with location component s is a reachable state of the transition system S_A. The input to the reachability problem consists of a timed automaton A and a set $L^F \subseteq L$ of *target* locations of A. The reachability problem is to determine whether or not some target location is reachable. Verification of safety requirements of real-time systems can be formulated as reachability problems for timed automata, as illustrated in the train-gate example. Since the transition system S_A of a timed automaton is infinite, our solution to the reachability problem involves construction of finite quotients.

Time-abstract transition system

Let A be a timed automaton. The transition system S_A has infinitely many states and infinitely many symbols. As a first step, we define another transition system whose transitions are labeled only with the symbols in Σ by hiding the labels denoting the time increments. For this purpose, whenever S_A contains a δ-labeled transition from state q to state q' and an a-labeled transition from q' to state q'', we add an a-labeled transition from q to q''. The resulting transition system is called *time-abstract* as it does not retain information about amount of time elapsed during transitions.

Formally, the time-abstract transition system of a timed automaton A, denoted U_A, is defined as follows. The state-space of U_A equals the state-space Q_A of S_A. The set of initial states of U_A equals the set of initial states of S_A. The set of labels of U_A equals the set Σ of labels of A. The transition relation of U_A is the relation $\Rightarrow$: for states q and q' and a label a, $q \overset{a}{\Rightarrow} q'$ iff there exists a state q'' and a time value $\delta \in \mathbb{R}$ such that $q \xrightarrow{\delta} q'' \xrightarrow{a} q'$ holds in the transition system S_A.

In the reachability problem for timed automata, we wish to determine reachability of target locations. Note that a location s of A is reachable iff some state with location s is reachable in the time-abstract transition system U_A. It follows that to solve reachability problems, we can consider the time-abstract transition system U_A instead of S_A.

Stable quotients

While the time-abstract transition system U_A has only finitely many labels, it still has infinitely many states. To address this problem, we consider equivalence relations over the state-space Q_A that group states together. An equivalence relation $\sim$ over the state-space Q_A of a timed automaton A is said to be *stable* iff whenever $q \sim u$ and $q \overset{a}{\Rightarrow} q'$, there exists a state u' such that $u \overset{a}{\Rightarrow} u'$ and $q' \sim u'$. In other words, a stable equivalence is a bisimulation of the time-abstract transition system.

Suppose A is a timed automaton, and $\sim$ is a stable partition of Q_A. Consider two equivalence classes π and π' of the relation $\sim$. Due to stability, π contains a state q such that $q \overset{a}{\Rightarrow} q'$ for some $q' \in \pi'$ iff for every q in π, $q \overset{a}{\Rightarrow} q'$ for some $q' \in \pi'$. This implies that to solve reachability problems, we need to keep track of only the equivalence class rather than individual states. The *quotient* of U_A with respect to a stable partition $\sim$ is the transition system $[U_A]_\sim$: states of $[U_A]_\sim$ are the equivalence classes of $\sim$, an equivalence class π is an initial state of $[U_A]_\sim$ if π contains an initial state of U_A, the set of labels is Σ, and $[U_A]_\sim$ contains an a-labeled transition from the equivalence class π to the class π' if for some $q \in \pi$ and $q' \in \pi'$, $q \overset{a}{\Rightarrow} q'$ holds in U_A.

To reduce the reachability problem (A, L^F) to a reachability problem over the quotient with respect to $\sim$, we need to ensure, apart from stability, that $\sim$ does not equate target states with non-target states. An equivalence relation $\sim$ is said to be L^F-sensitive, for a set $L^F \subseteq L$ of target locations, if whenever $(s, \nu) \sim (s', \nu')$, either both s and s' belong to L^F, or both s and s' do not belong to L^F.

Proposition 1 (Stable quotients) Let A be a timed automaton, $\sim$ be an equivalence relation over Q_A, and L^F be a set of locations of A such that $\sim$ is stable and L^F-sensitive. Then, a location in L^F is reachable iff there exists an equivalence class π of $\sim$ such that π is reachable in the quotient $[U_A]_\sim$ and π contains a state whose location is in L^F. ■

Consequently, to solve the reachability problem (A, L^F), we search for an equivalence relation $\sim$ that is stable, L^F-sensitive, and has only finitely many equivalence classes.

Restriction to integer constants

Recall that our definition of timed automata allows clock constraints which involve comparisons with rational constants. Given a timed automaton A, we can multiply each constant by the least common multiple of denominators of all the constants appearing in the clock constraints of A. This transformation leaves the answer to the reachability problem unchanged. Consequently, we can restrict ourselves to timed automata whose clock constraints involve only integer constants. Note that the largest constant in the transformed automaton is bounded by the product of the constants in the original automaton. It follows that the transformation causes at most a quadratic blow-up in the length of the encoding of the clock constraints.

Region equivalence

We define an equivalence relation on the state-space of an automaton that equates two states with the same location if they agree on the integral parts of all clock values and on the ordering of the fractional parts of all clock values. The integral parts of the clock values are needed to determine whether or not a particular clock constraint is met, whereas the ordering of the fractional parts is needed to decide which clock will change its integral part first. For example, if two clocks x and y are between 0 and 1 in a state, then a transition with clock constraint $(x = 1)$ can be followed by a transition with clock constraint $(y = 1)$, depending on whether or not the current clock values satisfy $(x < y)$. The integral parts of clock values can get arbitrarily large. But if a clock x is never compared with a constant greater than c, then its actual value, once it exceeds c, is of no consequence in deciding the allowed switches.

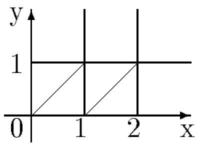
Now we formalize this notion. For any $\delta \in \mathbb{R}$, $fr(\delta)$ denotes the fractional part of δ, and $\lceil \delta \rceil$ denotes the integral part of δ; that is, $\delta = \lceil \delta \rceil + fr(\delta)$. For each clock $x \in X$, let c_x be the largest integer c such that x is compared with c in some clock constraint appearing in an invariant or a guard.

The equivalence relation $\cong$, called the *region equivalence*, is defined over the set of all clock interpretations for X. For two clock interpretations ν and ν', $\nu \cong \nu'$ iff all the following conditions hold:

1. For all $x \in X$, either $\lceil \nu(x) \rceil$ and $\lceil \nu'(x) \rceil$ are the same, or both $\nu(x)$ and $\nu'(x)$ exceed c_x.
2. For all $x, y \in X$ with $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, $fr(\nu(x)) \leq fr(\nu(y))$ iff $fr(\nu'(x)) \leq fr(\nu'(y))$.
3. For all $x \in X$ with $\nu(x) \leq c_x$, $fr(\nu(x)) = 0$ iff $fr(\nu'(x)) = 0$.

A *clock region* for A is an equivalence class of clock interpretations induced by $\cong$. We will use $[\nu]$ to denote the clock region to which ν belongs. Each region can be uniquely characterized by a (finite) set of clock constraints it satisfies. For example, consider a clock interpretation ν over two clocks with $\nu(x) = 0.3$ and $\nu(y) = 0.7$. Every clock interpretation in $[\nu]$ satisfies the constraint $(0 < x < y < 1)$, and we will represent this region by $[0 < x < y < 1]$.

The nature of the equivalence classes can be best understood through an example. Consider a timed transition table with two clocks x and y with $c_x = 2$ and $c_y = 1$. The clock regions are shown in Figure 5.



6 Corner points: e.g. [(0,1)]
14 Open line segments: e.g. $[0 < x = y < 1]$
8 Open regions: e.g. $[0 < x < y < 1]$

Figure 5: Clock regions

Note that there are only a finite number of regions. Also note that for a clock constraint φ of A, if $\nu \cong \nu'$ then ν satisfies φ iff ν' satisfies φ. We say that a clock region satisfies a clock constraint φ iff every clock interpretation in the region satisfies φ. Each region can be represented by specifying

(1) for every clock x, one clock constraint from the set

$$\{x = c \mid c = 0, 1, \ldots c_x\} \cup \{c - 1 < x < c \mid c = 1, \ldots c_x\} \cup \{x > c_x\},$$

(2) for every pair of clocks x and y such that $c - 1 < x < c$ and $d - 1 < y < d$ appear in (1) for some c, d, whether $fr(x)$ is less than, equal to, or greater than $fr(y)$.

By counting the number of possible combinations of equations of the above form, we conclude that the number of clock regions is bounded by $k! \cdot 2^k \cdot \Pi_{x \in X}(2c_x + 2)$, where k is the number of clocks. Thus, the number of clock regions is exponential in the encoding of the clock constraints.

Region automaton

Region equivalence relation $\cong$ over the clock interpretations is extended to an equivalence relation over the state-space by requiring equivalent states to have identical locations and region-equivalent clock interpretations: $(s, \nu) \cong (s', \nu')$ iff $s = s'$ and $\nu \cong \nu'$. The key property of region equivalence is its stability:

Proposition 2 (Stability) Region equivalence $\cong$ is stable. ■

For intuitive understanding of stability, let us consider the regions of Figure 5. Two states belonging to the same region satisfy the same set of guards of A, and hence, if a switch is possible from one state then the same switch is possible from the other. During the switch some clocks may be reset to 0. The effect of setting the clock x to 0 is projection onto the y-axis. Note that equivalent states project to equivalent states. Now consider the evolution of a state due to elapse of time. As time elapses, a state moves along the diagonally upwards direction since both clocks x and y increase at the same rate. For a given clock region, the sequence of regions encountered by such a translation along the diagonally upwards direction is the same irrespective of the choice of a state within the region.

The quotient $[U_A]_{\cong}$ of a timed automaton with respect to the region equivalence is called the *region automaton* of A, and is denoted $R(A)$. The number of equivalence classes of $\cong$ is finite, it is stable, and it is L^F-sensitive irrespective of the choice of the target locations. It follows that to solve the reachability problem (A, L^F), we can search the finite region automaton $R(A)$.

Consider the timed automaton A_0 shown in Figure 6. The alphabet is $\{a, b, c, d\}$. The corresponding region automaton $R(A_0)$ is shown in Figure 7. Only the regions reachable from the initial region

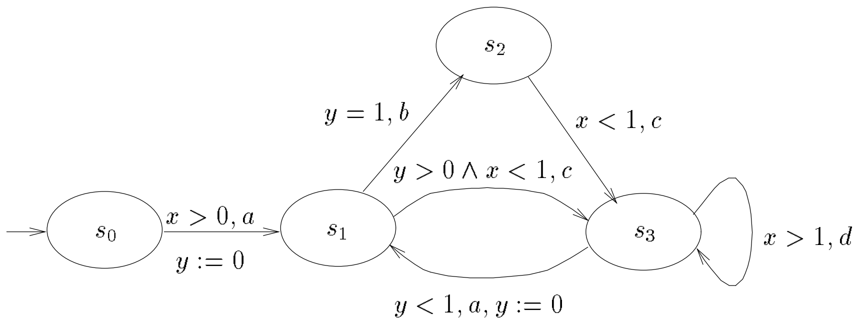


Figure 6: The automaton A_0

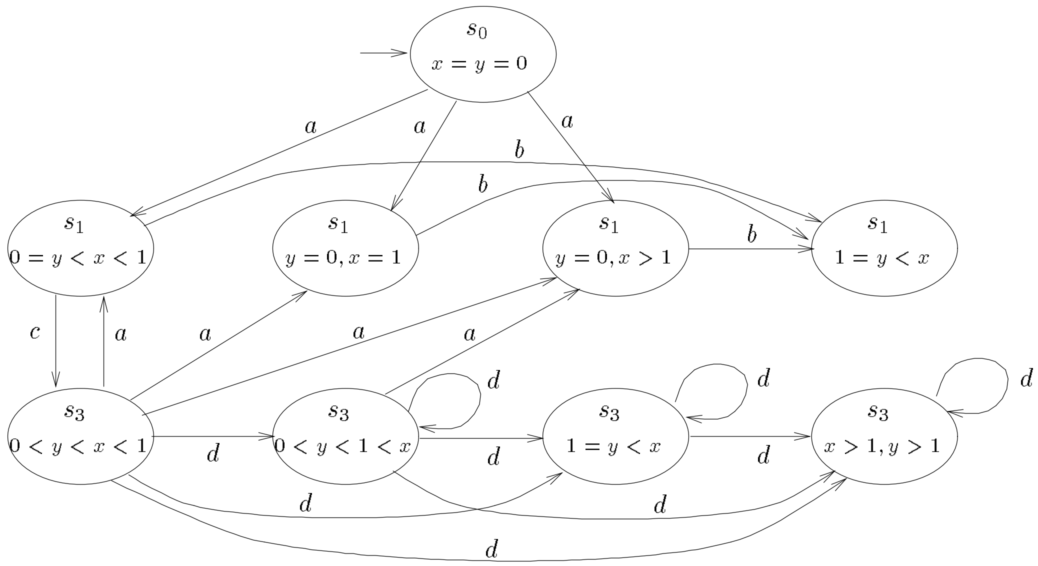


Figure 7: The region automaton $R(A_0)$

$\langle s_0, [x = y = 0]\rangle$ are shown. Note that $c_x = 1$ and $c_y = 1$. The timing constraints of the automaton ensure that the switch from s_2 to s_3 is never taken. The only reachable region with state component s_2 satisfies the constraints $[y = 1, x > 1]$, and this region has no outgoing edges. Thus the region automaton helps us in concluding that no switches can follow a b-labeled switch.

Complexity of reachability

Suppose every constant in the clock constraints of A is bounded by c. Then, the region automaton $R(A)$ has at most $n \cdot k! \cdot 2^k \cdot (2c+2)^k$ vertices, which is $n \cdot 2^{O(k\, log\, (kc))}$. To solve the reachability problem (A, L^F), we need to determine if the region automaton $R(A)$ has a reachable region with location in L^F. Such a search can be performed in time linear in the number of vertices and edges of the region automaton. Thus, the complexity of solving the reachability problem (A, L^F) is linear in the number of locations of A, exponential in the number of clocks, and exponential in the encoding of the constants. Technically, the reachability problem is PSPACE-complete. For membership in PSPACE, note that the reachability problem reduces to a search in an exponential automaton. PSPACE-hardness is established in [AD94]. In fact, in [CY91], it is established that both sources of complexity, the number of clocks and the magnitudes of the constants, render PSPACE-hardness independently of each other.

Theorem 3 (Complexity of Reachability) Let A be a timed automaton with n locations and k clocks, and suppose every constant in the clock constraints of A is bounded by c. Then, the reachability problem (A, L^F) can be solved in time $n \cdot 2^{O(k\, log\, (kc))}$. The reachability problem (A, L^F) is PSPACE-complete. ■

In practice, the input automaton for the reachability problem is a product of component automata. Thus, given component automata A_i, the solution to the reachability problem requires searching the region automaton $R(\|_i A_i)$. Observe that, while both the product construction and the region-automaton construction involve an exponential blow-up, the region automaton $R(\|_i A_i)$ is only singly-exponential in the descriptions of the component automata.

Remark 4 (Choice of timing constraints and decidability) The clock constraints in the enabling conditions and invariants of a timed automaton compare clocks with constants. Such constraints allow us to express (constant) lower and upper bounds on delays. For any generalization of the constraints, our analysis technique breaks down. In fact, if we allow constraints of the form $x = 2y$ (a special case of linear constraints over clocks), then the reachability problem becomes undecidable [AD94]. ■

Implementation

As noted earlier, the input to a verification problem consists of a set of component automata A_i, and the solution demands searching the region automaton $R(\|_i A_i)$. Let us first consider an enumerative implementation. A state of the region automaton $R(\|_i A_i)$ consists of locations of all the component automata, the integer parts of all the clocks, and the ordering of the fractional parts of all the clocks. Both the product construction and the region automaton construction is done on-the-fly, exploring successors of states of $R(\|_i A_i)$ only as needed.

Verification problems can be solved efficiently using symbolic search based on binary decision diagrams [BCD+92, McM93]. We sketch implementation of timed COSPAN [AK96] to illustrate the application of BDD-based search to timed reachability. The original COSPAN search engine can take an input file containing descriptions of coordinating automata described in the language S/R, and perform a symbolic search based on BDDs with many optimizations. In timed version, the input language for describing automata admits annotations to express timing constraints. Suppose the input program P consists of a collection of coordinating timed automata A_i. A preprocessor generates a new program P'. The automata in P' are described in original S/R without any timing annotations. For each A_i, let A'_i be the automaton without any timing annotations. The program P' consists of automata A'_i, together with the description of a monitor automaton A_R. Suppose $\|_i A_i$ has k clocks, and all the constants are bounded by c. The automaton A_R has $2k$ variables: k variables ranging over $0..c$ that keep track of the integral parts of the clocks, and k variables ranging over $1..k$ that give the ordering of the fractional parts. The update rules for these variables refer to the state-variables of the component automata. Searching the region automaton of $\|_i A_i$ is semantically equivalent to searching the product of $\|_i A_i$ with A_R. The details of the construction of the monitor automaton A_R are quite intricate, and heavily dependent on the language S/R. However, let us emphasize that the preprocessing step does not involve search, and can be performed in linear time. Following the preprocessing step, the search engine of COSPAN is used to perform the search on the input program P' using BDDs.

Zone automata

Let us revisit the region automaton of Figure 7. The initial region contains a single state $(s_0, x = y = 0)$, and has three successors with location s_1 corresponding to the clock regions $[y = 0 < x < 1]$, $[y = 0, x = 1]$, and $[y = 0, x > 1]$. One strategy is to collapse all these three regions together to obtain the union $[y = 0]$. Such convex unions of clock regions are called *clock zones*. A clock zone is a set of clock interpretations described by conjunction of constraints each of which puts a lower or upper bound on a clock or on difference of two clocks. Formally, the set of *clock zones* is generated by the grammar

$$\varphi \;:=\; x < c \mid x \leq c \mid c < x \mid c \leq x \mid x - y < c \mid x - y \leq c \mid \varphi_1 \wedge \varphi_2.$$

For a clock zone φ, the set of clock interpretations satisfying φ will also be denoted φ. If A has k clocks, then the set φ is a convex set in the k-dimensional euclidean space. Observe the following properties of clock zones.

1. Every clock region is a clock zone.

2. For a clock region π and a clock zone φ, either π is entirely contained in φ, or has an empty intersection with φ.

3. The intersection of two clock zones is a clock zone.

4. Every clock constraint used in the invariants of locations and guards of switches is a clock zone (follows from 1 and 3).

5. For two clock zones φ and ψ, if the union $\varphi \cup \psi$ is convex, then it is a clock zone. It follows that the set of clock zones coincides with the set of convex unions of regions.

The reachability analysis based on clock zones uses the following three operations on zones.

- For two clock zones φ and ψ, the intersection of the two zones is a zone, denoted $\varphi \wedge \psi$.

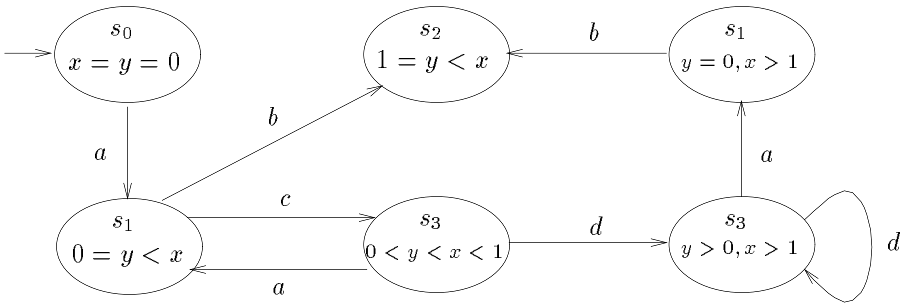


Figure 8: Reachable zone automaton

- For a clock zone φ, let $\varphi \Uparrow$ denote the set of clock interpretations $\nu + \delta$ for $\nu \in \varphi$ and $\delta \in \mathbb{R}$. Thus, $\varphi \Uparrow$ denotes the set of clock interpretations obtained by letting time elapse from some clock interpretation in φ. The set of clock zones is closed under this operation: for a clock zone φ, the set $\varphi \Uparrow$ is a clock zone.

- For a subset λ of clocks and a clock zone φ, let $\varphi[\lambda := 0]$ denote the set of clock interpretations $\nu[\lambda := 0]$ for $\nu \in \varphi$. Verify that $\nu[\lambda := 0]$ is a clock zone.

A *zone* is a pair (s, φ) for a location s and a clock zone φ. We build a transition system whose states are zones. Consider a zone (s, φ) and a switch $e = (s, a, \psi, \lambda, s')$ of A. Let $succ(\varphi, e)$ be the set of clock interpretations ν' such that for some $\nu \in \varphi$, the state (s', ν') can be reached from the state (s, ν) by letting time elapse and executing the switch e. That is, the set $(s', succ(\varphi, e))$ describes the successors of the zone (s, φ) under the switch e. To obtain the set $succ(\varphi, e)$, we (i) intersect φ with the invariant of s, (ii) let time ellapse using $\Uparrow$, (iii) take intersection with the invariant of s, (iv) take intersection with the guard ψ of e, and (v) reset the clocks in λ. The first and third steps ensure that the invariant is satisfied during elapse of time (since the invariant is convex, it suffices to ensure that the start and the final states satisfy the invariant). Thus,

$$succ(\varphi, e) \;=\; (((\varphi \wedge I(s)) \Uparrow) \wedge inv(s) \wedge \psi)[\lambda := 0]$$

Thus, the key property of the clock zones is closure under successors with respect to switches.

Proposition 4 (Zone Successor) For a clock zone φ and a switch e of a timed automaton A, the set $succ(\varphi, e)$ of clock interpretations is a clock zone. ■

A zone automaton is obtained by adding edges between zones (s, φ) and $(s', succ(\varphi, e))$. For a timed automaton A, the *zone automaton* $Z(A)$ is a transition system: states of $Z(A)$ are zones of A, for every initial location s of A, the zone $(s, [X := 0])$ is an initial location of $Z(A)$, and for every switch $e = (s, a, \psi, \lambda, s')$ of A and every clock zone φ, there is a transition $((s, \varphi), a, (s', succ(\varphi, e)))$.

Recall the automaton A_0 and its region automaton $R(A_0)$ of Figure 7. The reachable part of the zone automaton $Z(A_0)$ is shown in Figure 8. Note that, unlike the region automaton, in the zone automaton, each vertex has at most one successor per input symbol, and the number of vertices of $Z(A_0)$ is less than the number of vertices of $R(A_0)$.

Difference-bound matrices

Clock zones can be efficiently represented using matrices [Dil89]. Suppose the timed automaton A has k clocks, $x_1, \dots x_k$. Then a clock zone is represented by a $(k+1) \times (k+1)$ matrix D. For each i, the entry D_{i0} gives an upper bound on the clock x_i, and the entry D_{0i} gives a lower bound on the clock x_i. For every pair i, j, the entry D_{ij} gives an upper bound on the difference of the clocks x_i and x_j. To distinguish between a strict and a nonstrict bound (i.e. to distinguish between constraints such as $x < 2$ and $x \leq 2$), and allow for the possibility of absence of a bound, define the *bounds-domain* $\mathbb{D}$ to be $\mathbb{Z} \times \{0,1\} \cup \{\infty\}$. The constant ∞ denotes the absence of a bound, the bound $(c, 1)$, for $c \in \mathbb{Z}$, denotes the nonstrict bound $\leq c$, and the bound $(c, 0)$ denotes the strict bound $< c$. A *difference-bound matrix* (DBM) D is a $(k+1) \times (k+1)$ matrix D whose entries are elements from $\mathbb{D}$. A clock interpretation ν satisfies a DBM D iff for all $1 \leq i \leq k$, $x_i \leq D_{i0}$ and $-x_i \leq D_{0i}$, and for all $1 \leq i, j \leq k$, $x_i - x_j \leq D_{ij}$. Observe that every DBM represents a clock zone, and every clock zone is represented by some DBM.

As an example, consider the clock zone

$$(0 \leq x_1 < 2) \;\wedge\; (0 < x_2 < 1) \;\wedge\; (x_1 - x_2 \geq 0)$$

can be represented by the matrix D

	0	1	2
0	∞	(0,1)	(0,0)
1	(2,0)	∞	∞
2	(1,0)	(0,1)	∞

The DBM representation of a clock zone is not unique. In our example, there are many implied constraints that are not reflected in the matrix D. For instance, since $x_1 < 2$, we can conclude $x_1 - x_2 < 2$. Thus, the entry D_{12} can be updated from ∞ to $(2, 0)$ without changing the set of satisfying clock interpretations. The following DBM D' is equivalent to D:

	0	1	2
0	(0,1)	(0,1)	(0,0)
1	(2,0)	(0,1)	(2,0)
2	(1,0)	(0,1)	(0,1)

The matrix D' is obtained from the matrix by "tightening" all the constraints. Such a tightening is obtained by observing that sum of the upper bounds on the clock differences $x_i - x_j$ and $x_j - x_l$ is an upper bound on the difference $x_i - x_l$. Matrices like D' with tightest possible constraints are called *canonical*. To formalize this notion, we extend the operation of addition over integers to define addition over $\mathbb{D}$: adding ∞ to any element gives ∞, and $(i, j) + (i', j') = (i + i', j \wedge j')$. Similarly, we extend the comparison relation $<$ over integers to allow comparison of elements in $\mathbb{D}$: $(i, j) < \infty$, and $(i, j) < (i', j')$ iff $i < i'$, or $i = i'$ and $j < j'$.

The DBM D is *satisfiable* if it represents a nonempty clock zone. It can be established that D is unsatisfiable iff there exists a sequence $0 \leq i_1, i_2, \dots i_j \leq k$ of indices such that $D_{i_1 i_2} + D_{i_2 i_3} + \cdots + D_{i_j i_1} < (0, 1)$. The DBM D is said to be *canonical* iff for all $0 \leq i, j, l \leq k$, $D_{il} \leq D_{ij} + D_{jl}$. Every satisfiable DBM has an equivalent canonical DBM. We use canonical DBMs to represent clock zones. Given a DBM, using classical algorithms for computing all-pairs shortest paths, we check whether the DBM is satisfiable, and if so, convert it into a canonical form. Two canonical DBMs D

and D' are equivalent iff $D_{ij} = D'_{ij}$ for all $0 \leq i, j \leq k$. This test can be used during the search to determine if a zone has been visited earlier.

To implement reachability algorithm based on zones, given a DBM representation of a clock zone φ and a switch e, we need to compute the successor zone $succ(\varphi, e)$. For this purpose, we show how to obtain conjunction of two zones, and the representation of the zones $\psi \Uparrow$ and $\psi[\lambda := 0]$ from the representation of ψ.

- Intersection: Consider two canonical DBMs D and D'. Define the DBM D'' such that $D''_{ij} = min(D_{ij}, D'_{ij})$. Verify that D'' represents the intersection of the zones of D and D'. The matrix D'' need not be in canonical form, and we need to check if it is satisfiable, and if so, reduce it to canonical form.

- Time-elapse: Consider a DBM D, and suppose we wish to compute the matrix representation of $D \Uparrow$. The set $D \Uparrow$ contains all clock interpretations obtained from clock interpretations in D by letting time elapse. Note that the upper bounds on individual clocks are no longer valid when time elapses, while the lower bounds on individual clocks and the upper bounds on clock differences stay unchanged due to elapse of time. Consequently, given D, we update, for each $1 \leq i \leq k$, the entry D_{i0} to ∞, and canonicalize the resulting matrix to obtain $D \Uparrow$.

- Projection: Let us consider the effect of resetting a set λ of clocks to 0. Given a matrix D, to obtain $D[\lambda := 0]$, (i) for $x_i \in \lambda$, update D_{i0} and D_{0i} to $(0, 1)$, (ii) for $x_i, x_j \in \lambda$, update D_{ij} to $(0, 1)$, and (iii) for $x_i \in \lambda$ and $x_j \notin \lambda$, update D_{ij} to D_{0j} and D_{ji} to D_{j0}. As usual, the resulting matrix needs to be made canonical.

Theoretically, the number of zones is exponential in the number of regions, and thus, the zone automaton may be exponentially bigger than the region automaton. However, in practice, the zone automaton has fewer reachable vertices, and thus, leads to an improved performance. Furthermore, while the number of clock regions grows with the magnitudes of the constants used in the clock constraints, experience indicates that the number of reachable zones is relatively insensitive to the magnitudes of constants. As in case of region automata, the zone automaton is constructed on-the-fly.

Remark 5 (Dense vs discrete time) Our choice of time domain is $\mathbb{R}$, the set of nonnegative real numbers. Alternatively, we could choose $\mathbb{Q}$, the set of rational numbers, and all of the results stay unchanged. The key property of the time domain, in our context, is its denseness, which implies that arbitrarily many events can happen at different times in any interval of nonzero length. On the other hand, if we choose $\mathbb{N}$, the set of nonnegative integers, to model time, we have a discrete-time model, and the flavor of the analysis problems changes quite a bit. In the dense-time model, reachability for timed automata is PSPACE, while universality is undecidable; in the discrete-time case, reachability for timed automata is still PSPACE, while universality is EXPSPACE. We believe that discrete-time models, while appropriate for scheduling applications, are inappropriate for modeling asynchronous applications such as asynchronous circuits. For verification of real-time systems using discrete-time models, see, for instance, [EMSS90, CC94]. In [HMP92], it is established that under certain restrictions the timed reachability problem has the same answer irrespective of whether the time domain is $\mathbb{N}$ or $\mathbb{R}$. ■

Remark 6 (Minimization) Suppose we wish to explicitly construct a representation of the state-space of a timed automaton. Then, instead of building the region or the zone automaton, we can employ a minimization algorithm that constructs the coarsest stable refinement of a given initial partition by refining it as needed [ACH+92, YL93, KL94, TY96]. ■

3 Automata-Theoretic Verification

In the last section, we studied solutions to the reachability problem for timed automata. This is adequate to check *safety* properties of real-time systems. To verify *liveness* properties such as "if a request occurs infinitely often, so does the response" we need to consider nonterminating, infinite, executions. Specification and verification of both safety and liveness properties can be formulated in a uniform and elegant way using an automata-theoretic approach.

3.1 Verification via Automata Emptiness

In the *linear time model,* it is assumed that an execution can be completely modeled as a sequence of states or system events, called a *trace*. The behavior of the system is a set of such traces. Since a set of sequences is a formal language, this leads naturally to the use of automata for the specification and verification of systems. The more familiar definition of a formal language is as a set of finite words over some given (finite) alphabet. As opposed to this, an ω-language consists of infinite words. Thus an ω-language over a finite alphabet Σ is a subset of Σ^ω — the set of all infinite words over Σ. ω-automata provide a finite representation for certain types of ω-languages. An ω-automaton is essentially the same as a nondeterministic finite-state automaton, but with the acceptance condition modified suitably so as to handle infinite input words. We refer the reader to [Tho90] for a summary of the theory of ω-regular languages, and to [VW86, Kur94] for its application to verification.

Timed languages

To introduce time in the automata-theoretic framework, we begin by defining *timed words*–infinite sequences in which a real-valued time of occurrence is associated with each symbol. A *time sequence* $\overline{\tau} = \tau_1\tau_2\cdots$ is an infinite sequence of time values $\tau_i \in \mathbb{R}$, satisfying the following constraints:

1. *Monotonicity:* $\overline{\tau}$ increases monotonically; that is, $\tau_i \leq \tau_{i+1}$ for all $i \geq 1$.
2. *Progress (divergence):* For every $t \in \mathbb{R}$, there is some $i \geq 1$ such that $\tau_i > t$.

A *timed word* over an alphabet Σ is a pair $(\overline{\sigma}, \overline{\tau})$ consisting of an infinite word $\overline{\sigma} = \sigma_1\sigma_2\ldots$ over Σ and a time sequence $\overline{\tau}$. A *timed language* T over Σ is a set of timed words over Σ. If each symbol σ_i is interpreted to denote an event occurrence then the corresponding component τ_i is interpreted as the time of occurrence of σ_i. The progress requirement ensures that we disallow infinitely many events to occur within a finite interval of time. For example, define a timed language T_1 to consist of all timed words in which a and b alternate, and for the successive pairs of a and b, the time difference between a and b keeps increasing:

$$T_1 \;=\; \{((ab)^\omega, \overline{\tau}) \mid \text{ for } i \geq 1,\ ((\tau_{2i} - \tau_{2i-1}) < (\tau_{2i+2} - \tau_{2i+1}))\}.$$

The language-theoretic operations such as intersection, union, complementation are defined for timed languages as usual. In addition, we define the *Untime* operation which discards the time values associated with the symbols, that is, it considers the projection of a timed word $(\overline{\sigma}, \overline{\tau})$ on the first component: for a timed language T over Σ, $\mathit{Untime}(T)$ is the ω-language consisting of words $\overline{\sigma}$ such that $(\overline{\sigma}, \overline{\tau}) \in T$ for some time sequence $\overline{\tau}$. For instance, $\mathit{Untime}(T_1)$ consists of the single word $(ab)^\omega$.

Timed Büchi automata

In Section 1, we studied timed transition tables as an operational model for real-time systems. The same definition can be used as an acceptor that reads timed words. Let A be a timed automaton over the alphabet Σ. The automaton starts in an initial state. To read a timed word $(\overline{\sigma}, \overline{\tau})$, at every step $i \geq 1$, the automaton must let time elapse equal to the difference $\tau_i - \tau_{i-1}$ (let $\tau_0 = 0$), and then execute some σ_i-labeled location-switch. For instance, the timed automaton of Figure 1 can successfully read a timed word $(\overline{\sigma}, \overline{\tau})$ if $\overline{\sigma} = (ab)^\omega$ and $1 \leq \tau_{2i} - \tau_{2i-1} \leq 2$ for all $i \geq 1$. Formally, the *run* of the automaton A over the timed word $(\overline{\sigma}, \overline{\tau})$ is an infinite sequence r:

$$q_0 \xrightarrow{\tau_1} q_1' \xrightarrow{\sigma_1} q_1 \xrightarrow{\tau_2-\tau_1} q_2' \xrightarrow{\sigma_2} q_2 \xrightarrow{\tau_3-\tau_2} q_3' \xrightarrow{\sigma_3} q_3 \rightarrow \cdots \xrightarrow{\tau_i-\tau_{i-1}} q_i' \xrightarrow{\sigma_i} q_i \rightarrow \cdots$$

The run r is *initialized* if q_0 is an initial state of A. The set $\mathit{inf}(r) \subseteq L$ consists of those locations s of A for which s is the location-component of q_i for infinitely many indices i.

Different types of timed ω-automata can be defined by adding acceptance conditions to the definition of timed automata. We will use Büchi acceptance. A Büchi condition consists of a set of locations, and requires accepting runs to visit one of these locations infinitely often. Büchi conditions are useful to specify weak-fairness requirement for resolution of choice in modeling of nondeterministic systems.

A *timed Büchi automaton* (TBA) consists of a timed automaton $A = \langle L, L^0, \Sigma, X, I, E\rangle$ and a set $L^F \subseteq L$ of accepting locations of A. A run r of a TBA over a timed word $(\overline{\sigma}, \overline{\tau})$ is an *accepting run* iff the intersection $\mathit{inf}(r) \cap L^F$ is nonempty. For a TBA A, the language $T(A)$ of timed words it accepts is defined to be the set

$$\{(\overline{\sigma}, \overline{\tau}) \mid A \text{ has an accepting initialized run over } (\overline{\sigma}, \overline{\tau})\}.$$

For example, consider the automaton of Figure 2, and add the Büchi acceptance condition $\{s_0\}$. The accepting condition requires the location s_0 to be visited infinitely often. This rules out runs that loop at s_3 forever.

Remark 7 (Implicit liveness) Implicit in the definition of a timed automaton and its timed language, there are already two forms of liveness. First, since we require timed words to contain infinitely many symbols, every run must contain infinitely many switches. For instance, the automaton of Figure 1 cannot stay in the initial location s forever even though its invariant permits so (if we interpret a as input, we would like to allow the possibility of a never occurring, which can be modeled by adding a self-loop on location s labeled with an idling event). Second, the invariants of locations, together with the divergence of time sequences, ensure progress. For instance, the automaton of Figure 1 cannot stay in the location s' forever even if add a self-loop on s' labeled with an idling event. In fact, one can argue that adding explicit Büchi conditions, while useful for nondeterministic systems, is not needed for timed systems. However, let us note that accepting conditions are useful to specify *requirements*, and thus, timed Büchi automata provide a uniform framework for both system and its specification. ■

The product construction for timed automata is modified to obtain a product construction for timed Büchi automata so that a complex system can be defined as a product of TBAs. Let A_1 and A_2 be two timed Büchi automata. The location of the product, besides the locations of the component automata, contains a counter for cycling through the accepting sets of the component automata. The counter ranges over $\{0, 1, 2\}$, and is initially 0. It is updated from 0 to 1 when A_1

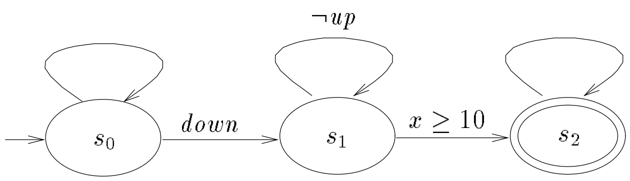


Figure 9: Violation of the bounded response property

executes a switch to some location in its accepting set L_1^F, from 1 to 2 when A_2 executes a switch to some location in its accepting set L_2^F, and is reset to 0 on the subsequent switch. If the counter equals 2 infinitely often, we can conclude that both automata execute switches to their respective accepting locations infinitely often. The details can be found in [AD94].

Specification and verification

In the automata-theoretic approach, a system is modeled as a timed Büchi automaton A, where A is typically a product of automata modeling system components. Verification corresponds to asking questions regarding the timed language $T(A)$. For instance, in the railroad controller example, T(GRC) consists of timed words over the alphabet {*approach, exit, in, out, lower, raise*}. Suppose we wish to prove the *bounded response property* that when the gate goes down, it is guaranteed to open within 10 units. This can be viewed as a requirement of the timed language T(GRC): in every timed word, the event *down* must be followed by an event *up* within 10 units. This property can be specified by a timed Büchi automaton. For ease of analysis, we require the designer to specify the complement of the desired property. Thus, the specification would be a TBA whose timed language is the set of timed words that *violate* the requirement. The specification, then, consists timed words $(\overline{\sigma}, \overline{\tau})$ such that for some i, $\sigma_i = down$, and for all $j > i$ such that $\tau_j - \tau_i < 10$, $\sigma_j \neq up$. The automaton corresponding to this specification is shown in Figure 9. The specification automaton starts in the initial location s_0. In the locations s_0 and s_2, it can synchronize with other components on any event, but in location s_1 it does not allow the event *up*. The accepting location is s_2, and a run leading to s_2 must contain the event *down* with no *up* for subsequent 10 units. It follows that the language of the product of GRC with the specification automaton is empty iff GRC satisfies the bounded response property.

In summary, the input to the *automata-theoretic timing verification problem* consists of a collection of timed Büchi automata A_i modeling the system components and the specification TBA A_S that accepts undesirable timed words. The verification problem is to determine whether the intersection of the timed languages $T(\|_i A_i)$ and $T(A_S)$ is empty.

Untiming

Consider the timing verification problem $(\|_i A_i, A_S)$. The verification problem corresponds to checking whether the timed language of the product of $\|_i A_i$ with A_S is empty. Since we already know how to construct products of TBAs, it suffices to find a decision procedure to check emptiness of the timed language of a single timed Büchi automaton. Consider a TBA A. First, observe that $T(A)$ is empty iff *Untime*$(T(A))$ is empty.

For a timed automaton A, its region automaton can be used to recognize $Untime(T(A))$. For this purpose, let us establish a correspondence between the runs of A and the runs (i.e. infinite paths) of $R(A)$. For a run r of A of the form

$$q_0 \xrightarrow{\tau_1} q_1' \xrightarrow{\sigma_1} q_1 \xrightarrow{\tau_2-\tau_1} q_2' \xrightarrow{\sigma_2} q_2 \xrightarrow{\tau_3-\tau_2} q_3' \xrightarrow{\sigma_3} q_3 \rightarrow \cdots \xrightarrow{\tau_i-\tau_{i-1}} q_i' \xrightarrow{\sigma_i} q_i \rightarrow \cdots$$

define its projection $[r]$ to be the sequence

$$[q_0] \xrightarrow{\sigma_1} [q_1] \xrightarrow{\sigma_2} [q_2] \xrightarrow{\sigma_3} [q_3] \rightarrow \cdots \xrightarrow{\sigma_i} [q_i] \rightarrow \cdots$$

where, for a state q, $[q]$ denotes the region it belongs to. From the definition of the edge relation for $R(A)$, it follows that $[r]$ is a run of $R(A)$. Since time progresses without bound along r, every clock $x \in X$ is either reset infinitely often, or from a certain time onwards it increases without bound. Hence, for all $x \in X$, for infinitely many $i \geq 0$, $[q_i]$ satisfies the clock constraint $(x = 0) \vee (x > c_x)$. This prompts the following definition: an infinite sequence $\pi_0\pi_1\ldots$ of regions is *progressive* iff for each clock $x \in X$, there are infinitely many $i \geq 0$ such that π_i satisfies $(x = 0) \vee (x > c_x)$. The correspondence between the runs of A and the runs of $R(A)$ can be made precise now (see [AD94] for a proof):

Proposition 5 An infinite sequence r of regions is a progressive run of the region automaton $R(A)$ over a word $\overline{\sigma}$ iff there exists a time sequence $\overline{\tau}$ and a run r' of A over $(\overline{\sigma}, \overline{\tau})$ such that r equals $[r']$. ∎

Consider the region automaton $R(A_0)$ of Figure 7. Every run r of $R(A_0)$ has a suffix of one of the following four forms: (i) the automaton cycles between the regions $(s_1, [y = 0 < x < 1])$ and $(s_3, [0 < y < x < 1])$, (ii) the automaton stays in the region $(s_3, [0 < y < 1 < x])$ using the self-loop, (iii) the automaton stays in the region $(s_3, [y = 0, x > 1])$ using the self-loop, or (iv) the automaton stays in the region $(s_3, [x > 1, y > 1])$. Only the case (iv) corresponds to the progressive runs. For runs of type (i), even though y gets reset infinitely often, the value of x is always less than 1. For runs of type (ii) or (iii), even though the value of x is not bounded, the clock y is reset only finitely often, and yet, its value is bounded. Thus every progressive run of A_0 corresponds to a run of $R(A_0)$ of type (iv).

Consequently, we need to add acceptance conditions to the region automaton so that only progressive runs satisfy the accepting conditions, and an accepting location of A is visited infinitely often. For the region automaton $R(A_0)$ of Figure 7, since all states of A_0 are accepting, from the description of the progressive runs, it follows that $R(A_0)$ can be changed to a Büchi automaton by choosing the accepting set to consist of a single region $\langle s_3, [x > 1, y > 1] \rangle$. Consequently, $Untime(T(A_0))$ equals $(ac)^+ d^\omega$. This leads to the main theorem for timed automata: the untimed language of a timed automaton is ω-regular.

Proposition 6 (From timed regular to ω-regular) Given a TBA A, there exists a Büchi automaton over Σ which accepts $Untime(T(A))$. ∎

Solving timing verification

For a timed Büchi automaton A, the language $T(A)$ is nonempty iff the region automaton $R(A)$ has a reachable cycle that contains an accepting location of A, and for every clock $x \in X$, contains a region satisfying $(x = 0)$ or a region satisfying $(x > c_x)$. Such a search can be performed in time linear in the size of the region automaton using, for instance, the nested depth-first search

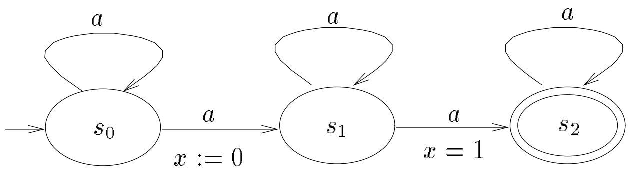


Figure 10: Noncomplementable automaton

algorithm of [CVWY92] to test Büchi emptiness. The complexity analysis of the timing verification problem is quite similar to the complexity analysis of the reachability problem. In particular, the complexity is linear in the number of locations of the product of all the automata, exponential in the total number of clocks, and exponential in the encoding of the clock constraints.

Theorem 7 (Complexity of timing verification) Let $(\|_i A_i, A_S)$ be an instance of the timing verification problem such that there are m component automata A_i. Suppose each of the $(m+1)$ automata has at most n locations and k clocks, and suppose every constant in the clock constraints of all the automata is bounded by c. Then, the timing verification problem $(\|_i A_i, A_S)$ can be solved in time $n^{m+1} \cdot 2^{O(km\, log\, (kcm))}$. The timing verification problem $(\|_i A_i, A_S)$ is PSPACE-complete. ■

The optimizations considered for the implementation of the solution to the reachability problem apply to the timing verification problem also. In particular, the construction of the region automaton of the product of the input automata is done on-the-fly, and the search can be done symbolically by introducing additional variables encoding clock constraints defining regions.

3.2 Theory of Timed Languages

In analogy with the class of languages accepted by Büchi automata, we call the class of timed languages accepted by TBAs timed regular languages: a timed language T is a *timed regular language* iff $T = T(A)$ for some TBA A. There is a well developed theory of timed regular languages. Here, we explain sample results, and their connection to analysis of real-time systems.

Closure properties

Since timed automata can have multiple initial locations, the class of timed regular languages is closed under union. The product construction for TBAs can be used to define intersection of timed regular languages, and thus, the class of timed regular languages is closed under intersection. However, the class is not closed under complementation. The language accepted by the automaton of Figure 10 over $\{a\}$ is

$$\{(a^\omega, \tau) \mid \text{for some } 1 \le i < j,\ (\tau_j = \tau_i + 1)\}.$$

The complement of this language cannot be characterized using a TBA. The complement needs to make sure that no pair of a's is separated by distance 1. Since there is no bound on the number of a's that can happen in a time period of length 1, keeping track of the times of all the a's within the past 1 time unit would require an unbounded number of clocks.

Proposition 8 (Closure properties) The class of timed regular languages is closed under union and intersection, but not under complementation. ■

The nonclosure under complementation depends on the denseness of the time domain.

Remark 8 (Alternative characterizations of timed regularity) The class of ω-regular languages is quite robust, and has many alternative characterizations (e.g. Büchi automata, Streett automata, the monadic second-order language S1S, ω-regular expressions). Obtaining equivalent characterizations of the class of timed regular languages seems much harder. Replacing Büchi accepting condition with the more general forms such as Streett acceptance or Muller acceptance, does not add expressiveness (i.e. the class of timed languages definable by timed Muller automata equals timed regular languages). Attempts have been made to define equivalent formulations using timed version of S1S [Wil94] and timed regular expressions [AMC97]. The effect of allowing ϵ-labeled (silent) switches, and related operations such as hiding is studied in [DGP97]. ■

Decision problems

As noted earlier, determining the emptiness of the timed language of a timed automaton reduces to searching for cycles in the region automaton, and is PSPACE-complete. Let us now consider the language inclusion problem for timed automata, that is, given two TBAs A_1 and A_2, we wish to determine if $T(A_1)$ is a subset of $T(A_2)$. When the specification automaton describes the desirable behaviors, then the verification problem corresponds to language inclusion. Let us note that for ordinary automata (or for Büchi automata) to test whether the language of one automaton is contained in another, we test the emptiness of the language of the product of the first automaton with the complement of the latter. This strategy cannot be used for TBAs, as it is not possible to automatically complement a TBA. In fact, there is no algorithm for testing whether the language of one TBA is contained in another. This is because the language inclusion problem, and even the weaker universality problem (i.e. whether a timed automaton accepts all timed words over its alphabet), is undecidable (see [AD94] for a proof via reduction from the halting problem of 2-counter machines).

Theorem 9 (Decision problems) The emptiness problem for timed Büchi automata is PSPACE-complete. The universality, language-inclusion, and language-equivalence problems for timed Büchi automata are undecidable. ■

Remark 9 (Finite timed words) Instead of considering languages of *infinite* timed words, we can consider language of *finite* timed words. Such finitary timed languages are adequate for reasoning about safety properties of real-time systems. If both the system and the specification are defined as finitary timed languages, then verification corresponds to checking reachability, rather than cycle detection, in the region automaton of the product. It should be noted, however, that negative results concerning languages of timed automata, such as PSPACE-hardness of emptiness and undecidability of universality, continue to hold in the finitary version. ■

Remark 10 (Timed bisimulation and timed simulation) While timed language equivalence is undecidable, stronger equivalences such as timed bisimulation and timed simulation are decidable. For a timed automaton A, a *timed bisimulation* is an equivalence relation $\sim$ on the state-space Q_A such that whenever $q_1 \sim q_2$, (1) if $q_1 \xrightarrow{a} q_1'$ for $a \in \Sigma \cup \mathbb{R}$, then there exists q_2' with $q_2 \xrightarrow{a} q_2'$ and $q_1' \sim q_2'$, and (2) if $q_2 \xrightarrow{a} q_2'$ for $a \in \Sigma \cup \mathbb{R}$, then there exists q_1' with $q_1 \xrightarrow{a} q_1'$ and $q_1' \sim q_2'$. While the number of equivalence classes of the maximal timed bisimulation relation is infinite,

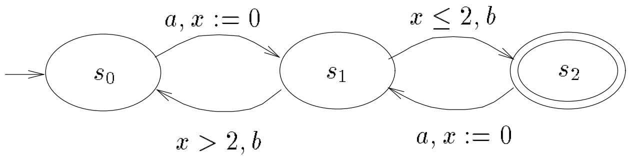


Figure 11: A deterministic timed Büchi automaton

the problem of deciding whether there exists a timed bisimulation that relates two specified initial states is, surprisingly, decidable [Č92] (the algorithm involves analysis of the region automaton of the product space $Q(A) \times Q(A)$). The same proof technique is useful to obtain algorithms for checking existence of timed simulation [TAKB96] (timed simulation relations are useful for establishing refinement between descriptions at different levels of abstractions). The complexity of deciding timed (bi)simulation is EXPTIME. A hierarchy of approximations to timed bisimulation relation can be defined on the basis of the *number* of clocks that an observer must use to distinguish between two timed automata [ACH94]. The impact of the *precision* of the observer's clocks on the distinguishing ability is studied in [LW93]. ■

Deterministic automata

The language inclusion problem is solvable if we use deterministic TBAs as specification automata. Recall that in the untimed case a deterministic automaton has a single initial location, and from each location, given the next input symbol, the next location is uniquely determined. We want a similar criterion for determinism for timed automata: given a location and the next input symbol *along with its time of occurrence*, the state after the next switch should be uniquely determined. So we allow multiple switches starting at the same location with the same symbol, but require their clock constraints to be *mutually exclusive* so that at any time only one of these switches is enabled. A timed automaton is called *deterministic* iff

1. it has only one initial location, and

2. for all $s \in L$, for all $a \in \Sigma$, for every pair of switches of the form $\langle s, a, \varphi_1, -, - \rangle$ and $\langle s, a, \varphi_2, -, - \rangle$, the clock constraints φ_1 and φ_2 are mutually exclusive (i.e., $\varphi_1 \wedge \varphi_2$ is unsatisfiable).

The automaton of Figure 11 is deterministic. The two b-labeled switches out of the location s_1 have mutually exclusive enabling conditions. By declaring the location s_2 to be the only accepting state, we can ensure that in every accepted timed word $((ab)^\omega, \overline{\tau})$, for infinitely many $i \geq 1$, $\tau_{2i} - \tau_{2i-1} \leq 1$.

Deterministic timed automata can be easily complemented because a deterministic timed automaton has at most one run over a given timed word. The algorithm for checking emptiness can be used to test whether the language of one TBA is included in the language of a deterministic TBA. More details regarding deterministic TBAs can be found in [AD94]. It is worth noting that there is no deterministic timed Büchi automaton whose language equals the language of the nondeterministic automaton of Figure 10.

Remark 11 (Event-clock automata) In [AFH97], a determinizable class of timed automata is obtained by restricting the use of clocks. The clocks of an *event-clock automaton* have a fixed, predefined association with the symbols of the input alphabet. The *event-recording clock* of the input symbol a is a history variable whose value always equals the time of the last occurrence of a relative to the current time; the *event-predicting clock* of a is a prophecy variable whose value always equals the time of the next occurrence of a relative to the current time (if no such occurrence exists, then the clock value is undefined). Thus, unlike a timed automaton, an event-clock automaton does not control the reassignments of its clocks, and, at each input symbol, all clock values of the automaton are determined solely by the input word. This property allows the determinization of event-clock automata, which, in turn, leads to a complementation procedure. Indeed, the class of event-clock automata is closed under all boolean operations (timed automata are not closed under complement), and the language-inclusion problem is decidable (PSPACE-complete) for event-clock automata. ■

4 Tools and Applications

A variety of tools exist for specification and verification of real-time systems. We briefly discuss three that are most closely related to the approach discussed in this paper.

Timed COSPAN

The tool COSPAN (see [HHK96] for an overview) is an automata-based modeling and analysis tool developed at Bell Labs. The real-time extension of COSPAN is described in [AK96]. The analysis of timing constraints can be done either using the region automaton or using the zone automaton, and the search can be performed either by an on-the-fly enumerative routine or by a BDD-based symbolic routine. Two approximations are supported as heuristic improvements. First, instead of analyzing all the timing constraints at once, they are added incrementally, as needed, in an automatic way guided by the results of the previous iterations [AIKY95]. Second, the underlying continuous semantics is approximated, in a conservative way, by the integers, and the former is used only when necessary. A recent version of timed COSPAN supports compositional refinement checking of timed automata using homomorphisms [TAKB96].

KRONOS

The tool KRONOS, developed at VERIMAG, supports analysis of a set of communicating timed automata [DOTY96]. The analysis can be performed by searching the zone automaton by an enumerative or a symbolic routine. The tool supports model checking of the branching real-time temporal logic TCTL, and interfaces to a variety of process-algebraic notations such as ET-LOTOS. Additional heuristics include a variety of minimization algorithms to compute reduced state-space [TY96]. KRONOS is available publicly at `http://www.imag.fr/VERIMAG/PEOPLE/Sergio.Yovine/kronos/`.

UPPAAL

The UPPAAL toolkit is developed in collaboration between Aalborg University, Denmark and Uppsala University, Sweden [LPY97]. Safety and bounded liveness properties of communicating timed automata are checked by an on-the-fly reachability analysis of the zone automaton.

Compositional techniques are used to reduce the search space [LPY95]. The tool supports a well-developed graphical user interface and features such as simulation. UPPAAL is available publicly at `http://www.docs.uu.se/docs/rtmv/uppaal/`.

Applications

The methodology described in this paper is suitable for finding logical errors in communication protocols and asynchronous circuits. Examples of analyzed protocols include Philips audio transmission protocol, carrier-sense multiple-access with collision detection, and Bang-Olufsen audio/video protocol (a detailed description of these and other case studies can be obtained from the homepages of KRONOS or UPPAAL). The application of COSPAN to verification of the asynchronous communication on the STARI chip is reported in [TB97], and to a scheduling problem in telecommunication software is reported in [AJKO97].

5 Discussion

We conclude by brief descriptions of related topics.

Linear real-time temporal logics Linear temporal logic (LTL) [Pnu77] is a popular formalism for writing requirements regarding computations of reactive systems. A variety of real-time extensions of LTL have been proposed for writing requirements of real-time systems [Ost90, Koy90, AH94, AFH96]. In particular, the real-time temporal logic *Metric Interval Temporal Logic* (MITL) admits temporal connectives such as *always*, *eventually*, and *until*, subscripted with intervals. A typical bounded-response requirement that "every request p must be followed by a response q within 3 time units" is expressed by the MITL formula

$$\Box(\ p \rightarrow \Diamond_{\leq 3}\ q).$$

To verify whether a real-time system modeled as a timed automaton A satisfies its specification given as a MITL formula φ, the model checking algorithm constructs a timed automaton $A_{\neg\varphi}$ that accepts all timed words that violate φ, and checks whether the product of A with $A_{\neg\varphi}$ has a nonempty language [AFH96]. The definition of MITL requires the subscripting intervals to be nonsingular. In fact, admitting singular intervals as subscripts (e.g. formulas of the form $\Box(p \rightarrow \Diamond_{=1}\ q)$) makes translation from MITL to timed automata impossible, and the satisfiability and model checking problems for the resulting logic are undecidable.

Branching real-time temporal logics Many tools for symbolic model checking employ the branching-time logic CTL [CE81, QS82] as a specification language. The real-time logic *Timed Computation Tree Logic* (TCTL) [ACD93] allows temporal connectives of CTL to be subscripted with intervals. For instance, the bounded response property that "every request p must be followed by a response q within 3 time units" is expressed by the TCTL formula

$$\forall\Box(\ p \rightarrow \forall\Diamond_{\leq 3}\ q).$$

It turns out that two states that are region-equivalent satisfy the same set of TCTL-formulas. Consequently, given a timed automaton A and a TCTL-formula φ, the computation the set of states of A that satisfy φ, can be performed by a labeling algorithm that labels the vertices of the region automaton $R(A)$ with subformulas of φ starting with innermost subformulas [ACD93]. Alternatively, the symbolic model checking procedure computes the set of states satisfying each subformula by a fixpoint routine that manipulates boolean combinations of zone constraints [HNSY94].

Probabilistic models Probabilistic extensions of timed automata allow modeling constraints such as "the delay between the input event a and the output event b is distributed uniformly between 1 to 2 seconds" (cf. [ACD91]). With introduction of probabilities, the semantics of the verification question changes. Given a probabilistic timed automaton A and a specification automaton A_S that accepts the undesirable behaviors, verification corresponds to establishing that the probability that the run of the system A generates a word accepted by A_S is zero. A modification of the cycle detection algorithm on the region automaton of the product of A and A_S can solve this problem [ACD91].

Hybrid systems The model of timed automata has been extended so that continuous variables other than clocks, such as temperature and imperfect clocks, can be modeled. *Hybrid automata* are useful in modeling discrete controllers embedded within continuously changing environment. Verifying correctness of hybrid automata is computationally more expensive than of timed automata, but in simple cases, such as the railroad controller, it allows reasoning with parametric bounds. We refer the reader to [ACH+95] for an introduction to hybrid automata, and to [HHW95] for an introduction to the verifier HyTech.

Acknowledgements

My research on timed automata has been in collaboration with Costas Courcoubetis, David Dill, Tom Henzinger, Bob Kurshan, and many others.

References

[ACD91] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for probabilistic real-time systems. In *Automata, Languages and Programming: Proceedings of the 18th ICALP*, LNCS 510, pages 115–136. Springer-Verlag, 1991.

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D.L. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *Proceedings of the Third Conference on Concurrency Theory*, LNCS 630, pages 340–354. Springer-Verlag, 1992.

[ACH94] R. Alur, C. Courcoubetis, and T.A. Henzinger. The observational power of clocks. In *Proceedings of the Fifth Conference on Concurrency Theory*, LNCS 836, pages 162–177. Springer-Verlag, 1994.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AFH96] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[AFH97] R. Alur, L. Fix, and T.A. Henzinger. A determinizable class of timed automata. *Theoretical Computer Science*, 204, 1997. A preliminary version appears in *Proc. CAV'94*, LNCS 818, pp. 1–13.

[AH94] R. Alur and T.A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[AH97] R. Alur and T.A. Henzinger. Modularity for timed and hybrid systems. In *Proceedings of the Eighth Conference on Concurrency Theory*, LNCS 1243, pages 74–88. Springer-Verlag, 1997.

[AIKY95] R. Alur, A. Itai, R.P. Kurshan, and M. Yannakakis. Timing verification by successive approximation. *Information and Computation*, 118(1):142–157, 1995.

[AJKO97] R. Alur, L.J. Jagadeesan, J.J. Kott, and J.E. Von Olnhausen. Model-checking of real-time systems: a telecommunications application. In *Proceedings of International Conference on Software Engineering*, 1997.

[AK96] R. Alur and R.P. Kurshan. Timing analysis in COSPAN. In *Hybrid Systems III: Control and Verification*, LNCS 1066, pages 220–231. Springer-Verlag, 1996.

[AL91] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In *Real-Time: Theory in Practice, REX Workshop*, LNCS 600, pages 1–27. Springer-Verlag, 1991.

[AMC97] E. Asarin, O. Maler, and P. Caspi. A Kleene theorem for timed automata. In *Proceedings of the 12th IEEE Symposium on Logic in Computer Science*, pages 160–171, 1997.

[BCD+92] J.R. Burch, E.M. Clarke, D.L. Dill, L.J. Hwang, and K.L. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BST98] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *Compositionality – the significant difference*, LNCS. Springer-Verlag, 1998.

[CC94] S. Campos and E.M. Clarke. Real-time symbolic model checking for discrete time models. In *Theories and experiences for real-time system development*, AMAST series in computing, 1994.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, LNCS 131, pages 52–71. Springer-Verlag, 1981.

[CK96] E.M. Clarke and R.P. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[CY91] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In *Proceedings of the Third Workshop on Computer-Aided Verification*, LNCS 575, pages 399–409. Springer-Verlag, 1991.

[DGP97] V. Diekert, P. Gastin, and A. Petit. Removing ϵ-transitions in timed automata. In *Proceedings of the 14th Symposium on Theoretical Aspects of Computer Science*, LNCS 1200, pages 583–594. Springer-Verlag, 1997.

[Dil89] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 197–212. Springer–Verlag, 1989.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III: Verification and Control*, LNCS 1066, pages 208–219. Springer-Verlag, 1996.

[EMSS90] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Computer-Aided Verification, 2nd International Conference, CAV'90*, LNCS 531, pages 136–145. Springer-Verlag, 1990.

[GSSL94] R. Gawlick, R. Segala, J. Sogaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. In *Automata, Languages, and Programming, Proceedings of the 21st ICALP*, LNCS 820, pages 166–177. Springer-Verlag, 1994.

[HHK96] R. Hardin, Z. Har'El, and R.P. Kurshan. COSPAN. In *Proceedings of the Eighth International Conference on Computer Aided Verification*, LNCS 1102, pages 423–427. Springer-Verlag, 1996.

[HHW95] T.A. Henzinger, P. Ho, and H. Wong-Toi. HyTech: the next generation. In *TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1019, pages 41–71. Springer-Verlag, 1995.

[HMP92] T.A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *ICALP 92: Automata, Languages, and Programming*, LNCS 623, pages 545–558. Springer-Verlag, 1992.

[HMP94] T.A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112(2):273–337, 1994.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[JM87] F. Jahanian and A.K. Mok. A graph-theoretic approach for timing analysis and its implementation. *IEEE Transactions on Computers*, C-36(8):961–975, 1987.

[KL94] I. Kang and I. Lee. State minimization for concurrent system analysis based on state space exploration. In *Proceedings of the Conference On Computer Assurance*, pages 123–134, 1994.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *Journal of Real-Time Systems*, 2:255–299, 1990.

[Kur94] R.P. Kurshan. *Computer-aided Verification of Coordinating Processes: the automata-theoretic approach*. Princeton University Press, 1994.

[LA92] N.A. Lynch and H. Attiya. Using mappings to prove timing properties. *Distributed Computing*, 6:121–139, 1992.

[LPY95] K. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, 1995.

[LPY97] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1, 1997.

[LW93] K. Larsen and Y. Wang. Time abstracted bisimulation: Implicit specifications and decidability. In *Proceedings of Mathematical Foundations of Programming Semantics*, 1993.

[McM93] K. McMillan. *Symbolic model checking: an approach to the state explosion problem*. Kluwer Academic Publishers, 1993.

[MMT91] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In *Proceedings of Workshop on Theories of Concurrency*, 1991.

[Ost90] J. Ostroff. *Temporal Logic of Real-time Systems*. Research Studies Press, 1990.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–77, 1977.

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent programs in CESAR. In *Proceedings of the Fifth International Symposium on Programming*, LNCS 137, pages 195–220. Springer-Verlag, 1982.

[RR88] G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.

[TAKB96] S. Tasiran, R. Alur, R.P. Kurshan, and R.K. Brayton. Verifying abstractions of timed systems. In *Proceedings of the Seventh Conference on Concurrency Theory*, LNCS 1119, pages 546–562. Springer-Verlag, 1996.

[TB97] S. Tasiran and R. Brayton. STARI: a case study in compositional and hierarchical timing verification. In *Proceedings of the Ninth International Conference on Computer Aided Verification*, LNCS 1254, pages 191–201. Springer-Verlag, 1997.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science Publishers, 1990.

[TY96] S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulations. In *Proceedings of the Eighth International Conference on Computer Aided Verification*, LNCS 1102. Springer-Verlag, 1996.

[Č92] K. Čerāns. Decidability of bisimulation equivalence for parallel timer processes. In *Proceedings of the Fourth Workshop on Computer-Aided Verification*, LNCS 663, pages 302–315. Springer-Verlag, 1992.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First IEEE Symposium on Logic in Computer Science*, pages 332–344, 1986.

[Wil94] T. Wilke. Specifying state sequences in powerful decidable logics and timed automata. In *Proceedings of Third International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems*, LNCS 863, pages 694–715. Springer-Verlag, 1994.

[YL93] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In *Proceedings of the Fifth Conference on Computer-Aided Verification*, LNCS 697, pages 210–224. Springer-Verlag, 1993.