Timed Automata

Rajeev Alur*

Abstract. Model checking is emerging as a practical tool for automated debugging of complex reactive systems such as embedded controllers and network protocols (see [23] for a survey). 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* [7] were introduced as a formal notation to model the behavior of real-time systems. Its definition provides a simple 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 the theory of timed automata, and their role in specification and verification of real-time systems.

1 Modeling

Transition systems. We model discrete 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. 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 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, 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.

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 a simple example.

* Department of Computer and Information Science, University of Pennsylvania, and Bell Laboratories, Lucent Technologies. Email: alur@cis.upenn.edu. Supported in part by NSF CAREER award CCR-9734115 and by the DARPA grant NAG2-1214.

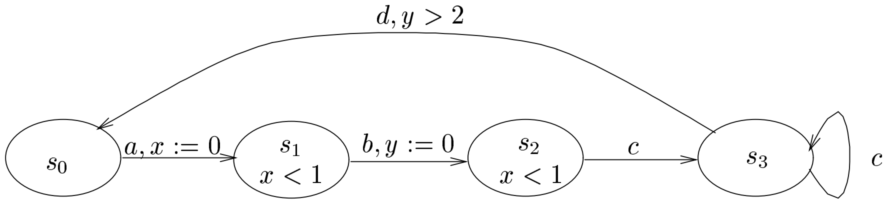


Fig. 1. A timed automaton with 2 clocks

Consider the timed automaton of Figure 1 with two clocks. 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 switch 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. 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.

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. For a set X of clocks, 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. 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 with some clock constraint in $\Phi(X)$, and
- $E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L$ is a set of switches. A switch $\langle s, a, \varphi, \lambda, s'\rangle$ represents an edge 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 change 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 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 1, 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

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

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

Remark 1 (Nonzenoness). 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 [1, 29, 11]. ■

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 has $\langle (s_1, s_2), a, \varphi_1 \wedge \varphi_2, \lambda_1 \cup \lambda_2, (s_1', s_2') \rangle$.
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 has $\langle (s, t), a, \varphi, \lambda, (s', t) \rangle$.
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 has $\langle (t, s), a, \varphi, \lambda, (t, s') \rangle$.

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 by synchronizing the switches with identical labels.

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 2. 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 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 $x > 2$. The computational problem in timing verification is to make such deductions by analyzing the timing constraints.

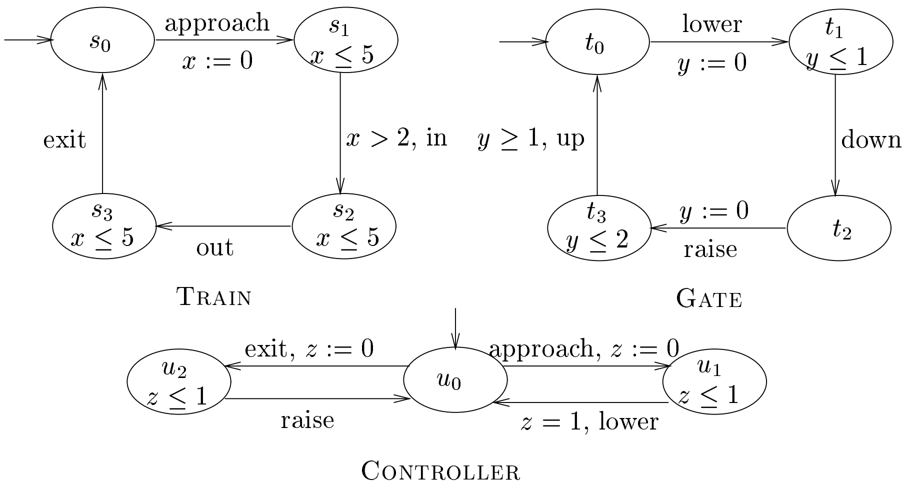


Fig. 2. Train-gate controller

Remark 2 (Compositionality). For communication between system components, many competing alternatives to the definition used in this paper exist. The choice of synchronization primitives is somewhat orthogonal to the problem of analysis of timing constraints, and the algorithmic techniques for timed automata can be applied to other models. 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 [44, 11, 43]. Issues pertaining to the impact of timing on synchronization are studied in [19]. ■

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. The transition system S_A of a timed automaton A has infinitely many states and infinitely many symbols. As a first step, we define another transition system, called the *time-abstract* transition system and denoted U_A, whose transitions are labeled only with the symbols in Σ by hiding the labels denoting the time increments. 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 $q \overset{\delta}{\rightarrow} q'' \overset{a}{\rightarrow} q'$ holds in the transition system S_A. In the reachability problem for timed automata, we wish to determine reachability of target locations. 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. An equivalence relation $\sim$ over the state-space Q_A is said to be *stable* iff whenever $q \sim u$ and $q \stackrel{a}{\Rightarrow} q'$, there exists a state u' such that $u \stackrel{a}{\Rightarrow} u'$ and $q' \sim u'$. 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 $q' \in \pi'$, $q \stackrel{a}{\Rightarrow} q'$ holds in U_A.

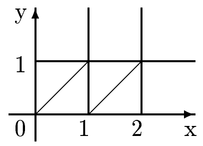
To reduce the reachability problem (A, L^F) to a reachability problem over the quotient with respect to $\sim$, we need to ensure, apart from stability, that $\sim$ does not equate target states with non-target states. An equivalence relation $\sim$ is said to be L^F-sensitive, for a set $L^F \subseteq L$ 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. 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.

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. Here, we are assuming that all clock constraints involve comparisons with integer constants (if the clock constraints involve rational constants, we can multiply each constant by the least common multiple of denominators of all the constants).

Now we formalize this notion. For any $\delta \in \mathbb{R}$, $fr(\delta)$ denotes the fractional part of δ, and $\lfloor \delta \rfloor$ denotes the integral part of δ; that is, $\delta = \lfloor \delta \rfloor + fr(\delta)$. 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 $\lfloor \nu(x) \rfloor$ and $\lfloor \nu'(x) \rfloor$ are the same, or both $\nu(x)$ and $\nu'(x)$ exceed c_x.
2. For all x, y 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 $\cong$. 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 3. Note that there are only a finite number of regions, at most $k! \cdot 2^k \cdot \Pi_{x \in X}(2c_x + 2)$, where k is the number of clocks. Thus, the number of clock regions is exponential in the encoding of the clock constraints.



6 Corner points: e.g. [(0,1)]

14 Open line segments: e.g. $[0 < x = y < 1]$

8 Open regions: e.g. $[0 < x < y < 1]$

Fig. 3. Clock regions

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. 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 $R(A)$.

Complexity of reachability. Reachability can be solved in time linear in the number of vertices and edges of the region automaton, which is linear in the number of locations, exponential in the number of clocks, and exponential in the encoding of the constants. Technically, the reachability problem is PSPACE-complete. In fact, in [24], 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.

Remark 3 (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 [7]. ■

Zone automata. One strategy to improve the region construction is to collapse regions by considering convex unions of clock regions. A *clock zone* φ 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. If A has k clocks, then the set φ is a convex set in the k-dimensional euclidean space.

The reachability analysis using zones uses the following three operations:

- For two clock zones φ and ψ, $\varphi \wedge \psi$ denotes the intersection of the two zones.
- For a clock zone φ, $\varphi \Uparrow$ denotes the set of interpretations $\nu + \delta$ for $\nu \in \varphi$ and $\delta \in \mathbb{R}$.
- For a subset λ of clocks and a clock zone φ, $\varphi[\lambda := 0]$ denotes the set of clock interpretations $\nu[\lambda := 0]$ for $\nu \in \varphi$.

A key property of the set of clock zones is closure under the above three operations. 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. The set $succ(\varphi, e)$ can be computed using the three operations on clock zones as follows:

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

Thus, clock zones are effectively closed under successors with respect to switches. 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 $((s, \varphi), a, (s', succ(\varphi, e)))$.

Difference-bound matrices. Clock zones can be efficiently represented using matrices [27]. Suppose the timed automaton A has k clocks, $x_1, \ldots x_k$. 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* $\mathbb{K}$ 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 $\mathbb{K}$. 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 as well as by the matrix D':

	Matrix D			Matrix D'		
	0	1	2	0	1	2
0	∞	(0,1)	(0,0)	(0, 1)	(0,1)	(0,0)
1	(2,0)	∞	∞	(2,0)	(0,1)	(2,0)
2	(1,0)	(0,1)	∞	(1,0)	(0,1)	(0,1)

Observe that there are many implied constraints that are not reflected in the matrix D, while the matrix D' is obtained from the matrix D 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 $x_i - x_l$ (for this purpose, the operations of $+$ and $<$ are extended to the domain $\mathbb{K}$ of bounds). Matrices like D' with tightest possible constraints are called *canonical*. The DBM D is *satisfiable* if it represents a nonempty clock zone. 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. The representation using canonical DBMs supports the required operations of conjunction, $\psi \Uparrow$, and $\psi[\lambda := 0]$ efficiently (cf. [27]).

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.

Implementation. The input to a verification problem consists of a set of component timed automata A_i, and the solution demands searching the region automaton $R(\|_i A_i)$ or $Z(\|_i A_i)$. The actual search can be performed by an on-the-fly enumerative engine or a BDD-based symbolic engine. We briefly sketch implementation of the search in timed COSPAN [15]. Suppose the input program P consists of a collection of coordinating timed automata A_i. For each A_i, let A'_i be the automaton without any timing annotations. A preprocessor generates a new program P' that consists of automata A'_i, together with the description of a monitor automaton A_R encoding the region construction or A_Z encoding the DBM-based zone construction. 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 automaton A_Z has $(k+1)^2$ variables ranging over $-c..c$ that keep track of the numerical entries in the DBM and $(k+1)^2$ boolean variables that keep track of the strictness bit for each matrix entry. 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, while searching the zone automaton of $\|_i A_i$ is semantically equivalent to searching the product of $\|_i A'_i$ with A_Z. Following the preprocessing step, the search engine of COSPAN is used to perform the search on the input program P' using BDDs or using on-the-fly enumerative search. Experience shows that for enumerative search the zone construction is preferable, while for symbolic search the region construction is preferable.

Remark 4 (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, [28, 21]. In [34], it is established that under certain restrictions the timed reachability problem has the same answer irrespective of choice between $\mathbb{N}$ and $\mathbb{R}$. ■

Remark 5 (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 [4, 54, 37, 50]. ■

Remark 6 (Alternative Symbolic Representations). There have been many attempts to combine BDD-based representation of discrete locations with DBM-based representation of zones. Sample approaches include encoding DBMs using BDDs with particular attention to bit patterns in the variable ordering [20], and variants of BDDs specifically designed to represent clock constraints [18]. ■

3 Discussion

We have summarized the basic techniques for analysis of timed automata (see also [41] for an introduction). We conclude by briefly discussing tools, applications, and theoretical results.

Tools. A variety of tools exist for specification and verification of real-time systems. We list three that are most closely related to the approach discussed in this paper. The tool *timed* COSPAN is is an automata-based modeling and analysis tool developed at Bell Labs (see [15, 13]). The tool KRONOS, developed at VERIMAG, supports model checking of branching-time requirements [25]. The UPPAAL toolkit is developed in collaboration between Aalborg University, Denmark and Uppsala University, Sweden [40] and allows checking of safety and bounded liveness properties. All these tools incorporate many additional heuristics for improving the performance.

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 [49], and to a scheduling problem in telecommunication software is reported in [14].

Automata-theoretic Verification. Reachability analysis discussed in Section 2 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 [52, 39, 7]. In this approach, a timed automaton, possibly with acceptance conditions (e.g. Büchi), is viewed as a generator of a *timed language* – a set of sequences in which a real-valued time of occurrence is associated with each symbol. Verification corresponds to queries about the timed language defined by the timed automaton modeling the system. If the query is given by a timed automaton that accepts *undesirable* behaviors, then verification question reduces to checking emptiness of the intersection, and can be solved in PSPACE. On the other hand, if the query is given by a timed automaton that accepts all behaviors satisfying the desired property, verification corresponds to testing inclusion of the two timed languages, and is undecidable in general [7]. Decidability of the language-inclusion problem can be ensured by requiring the specification automaton to be *deterministic*, or an *event-clock automaton*.

Since theory of regular (or ω-regular) languages finds many applications including modeling of discrete systems, many attempts have been made to develop a corresponding theory of timed languages. Timed languages defined by timed automata can be characterized using timed version of S1S [53], timed regular expressions [17], and timed temporal logics [36]. The complexity of different types of membership problems for timed automata is studied in [16]. Timed languages definable by timed automata are closed under union and intersection, but not under complementation. This has prompted identification of subclasses such as event-clock automata [9] with better closure properties.

Equivalence and Refinement Relations. While timed language equivalence for timed automata is undecidable, stronger equivalences such as timed bisimulation and 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 $q_1 \sim q_2$, 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 $q_1' \sim q_2'$. While the number of equivalence classes of the maximal timed bisimulation relation is infinite, the problem of deciding whether there exists a timed bisimulation that relates two specified initial states is, surprisingly, decidable [51] (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 [48] (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 [6]. The impact of the *precision* of the observer's clocks on the distinguishing ability is studied in [42].

Linear real-time temporal logics. Linear temporal logic (Ltl) [46] 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 [45, 38, 10, 8]. 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 [8]. 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. See [31] for a recent survey of real-time temporal logics.

Branching real-time temporal logics. Many tools for symbolic model checking employ the branching-time logic Ctl [22, 47] as a specification language. The real-time logic *Timed Computation Tree Logic* (Tctl) [3] 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 [3]. 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 [35].

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. [2]). 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 [2]. A similar approach works for verifying Tctl properties of a probabilistic timed automaton. However, if we introduce explicit probabilities in the requirements (e.g. event a will happen within time 2 with probability at least 0.5), then model checking algorithms are known only for a discrete model of time [26].

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. Verification of hybrid automata is undecidable in general. For the subclass of *rectangular automata*, analysis is possible via language-preserving translation to timed automata [33], and for the subclass of *linear hybrid automata*, analysis is possible based on symbolic fixpoint computation using polyhedra [12]. See [5] for an introduction to the theory, to [32] for an introduction to the tool HyTech, and to [30] for a survey.

Acknowledgements. My research on timed automata has been in collaboration with Costas Courcoubetis, David Dill, Tom Henzinger, Bob Kurshan, and many others. Many thanks to them, and to Salvatore La Torre for comments on this survey.

References

1. 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.
2. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems. In *Automata, Languages and Programming: Proc. of 18th ICALP*, LNCS 510, pages 115–136, 1991.
3. R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
4. R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *CONCUR '92*, LNCS 630, pages 340–354, 1992.
5. R. Alur, C. Courcoubetis, N. Halbwachs, T. 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.
6. R. Alur, C. Courcoubetis, and T. Henzinger. The observational power of clocks. In *CONCUR '94: Fifth Conference on Concurrency Theory*, LNCS 836, pages 162–177, 1994.
7. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
8. R. Alur, T. Feder, and T. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.
9. R. Alur, L. Fix, and T. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999.
10. R. Alur and T. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.
11. R. Alur and T. Henzinger. Modularity for timed and hybrid systems. In *CONCUR '97: Eighth Conference on Concurrency Theory*, LNCS 1243, pages 74–88, 1997.
12. R. Alur, T. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.
13. R. Alur, A. Itai, R. Kurshan, and M. Yannakakis. Timing verification by successive approximation. *Information and Computation*, 118(1):142–157, 1995.
14. R. Alur, L. Jagadeesan, J. Kott, and J. V. Olnhausen. Model-checking of real-time systems: a telecommunications application. In *Proc. of Intl. Conf. on Software Engineering*, 1997.
15. R. Alur and R. Kurshan. Timing analysis in COSPAN. In *Hybrid Systems III: Control and Verification*, LNCS 1066, pages 220–231. Springer-Verlag, 1996.
16. R. Alur, R. Kurshan, and M. Viswanathan. Membership problems for timed and hybrid automata. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*. 1998.
17. 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.
18. G. Behrmann, K. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In *Computer Aided Verification*, 1999.
19. S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *Compositionality – the significant difference*, LNCS. Springer-Verlag, 1998.
20. M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In *Computer Aided Verification*, LNCS 1254, pages 179–190. 1997.
21. S. Campos and E. Clarke. Real-time symbolic model checking for discrete time models. In *Theories and experiences for real-time system development*, AMAST series in computing, 1994.
22. E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, LNCS 131, pages 52–71, 1981.
23. E. Clarke and R. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.
24. C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In *Third Workshop on Computer-Aided Verification*, LNCS 575, pages 399–409, 1991.
25. 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.

26. L. de Alfaro. *Formal verification of probabilistic systems.* PhD thesis, Stanford University, 1997.
27. D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 197–212, 1989.
28. E. Emerson, A. Mok, A. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Computer-Aided Verification, 2nd International Conference*, LNCS 531, pages 136–145, 1990.
29. R. Gawlick, R. Segala, J. Sogaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. In *Proc. ICALP'94*, LNCS 820, pages 166–177, 1994.
30. T. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th IEEE Symposium on Logic in Computer Science*, pages 278–293, 1996.
31. T. Henzinger. It's about time: Real-time logics reviewed. In *CONCUR '98: Ninth International Conference on Concurrency Theory*, LNCS 1466, pages 439–454. 1998.
32. T. 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, 1995.
33. T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata. In *Proceedings of the 27th ACM Symposium on Theory of Computing*, pages 373–382, 1995.
34. T. 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.
35. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.
36. T. Henzinger, J. Raskin, and P. Schobbens. The regular real-time languages. In *ICALP 98: Automata, Languages, and Programming*, LNCS 1443, pages 580–593. 1997.
37. 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.
38. R. Koymans. Specifying real-time properties with metric temporal logic. *Journal of Real-Time Systems*, 2:255–299, 1990.
39. R. Kurshan. *Computer-aided Verification of Coordinating Processes: the automata-theoretic approach.* Princeton University Press, 1994.
40. K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1, 1997.
41. K. Larsen, B. Steffen, and C. Weise. Continuous modeling of real-time and hybrid systems: from concepts to tools. *Software Tools for Technology Transfer*, 1(2):64–85, 1997.
42. K. Larsen and Y. Wang. Time abstracted bisimulation: Implicit specifications and decidability. In *Proceedings of Mathematical Foundations of Programming Semantics*, 1993.
43. N. Lynch. *Distributed algorithms.* Morgan Kaufmann, 1996.
44. M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In *Proceedings of Workshop on Theories of Concurrency*, 1991.
45. J. Ostroff. *Temporal Logic of Real-time Systems.* Research Studies Press, 1990.
46. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–77, 1977.
47. J. Queille and J. Sifakis. Specification and verification of concurrent programs in CESAR. In *Proceedings of the Fifth Symposium on Programming*, LNCS 137, pages 195–220, 1982.
48. S. Tasiran, R. Alur, R. Kurshan, and R. Brayton. Verifying abstractions of timed systems. In *CONCUR '96*, LNCS 1119, pages 546–562, 1996.
49. S. Tasiran and R. Brayton. STARI: a case study in compositional and hierarchical timing verification. In *CAV'97*, LNCS 1254, pages 191–201, 1997.
50. S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulations. In *Proceedings of the Eighth Conference on Computer Aided Verification*, LNCS 1102, 1996.
51. K. Čerāns. Decidability of bisimulation equivalence for parallel timer processes. In *CAV'92*, LNCS 663, pages 302–315, 1992.
52. M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. of the First IEEE Symp. on Logic in Computer Science*, pages 332–344, 1986.
53. T. Wilke. Specifying state sequences in powerful decidable logics and timed automata. In *FTRTFT'94*, LNCS 863, pages 694–715. Springer-Verlag, 1994.
54. M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In *CAV'93*, LNCS 697, pages 210–224, 1993.