A Determinizable Class of Timed Automata*

Rajeev Alur[†] Limor Fix[‡] Thomas A. Henzinger[§]

Abstract. We introduce the class of *event-recording timed automata* (ERA). An event-recording automaton contains, for every event a, a clock that records the time of the last occurrence of a. The class ERA is, on one hand, expressive enough to model (finite) timed transition systems and, on the other hand, determinizable and closed under all boolean operations. As a result, the language inclusion problem is decidable for event-recording automata. We present a translation from timed transition systems to event-recording automata, which leads to an algorithm for checking if two timed transition systems have the same set of timed behaviors.

We also consider *event-predicting timed automata* (EPA), which contain clocks that predict the time of the next occurrence of an event. The class of *event-clock automata* (ECA), which contain both event-recording and event-predicting clocks, is a suitable specification language for real-time properties. We provide an algorithm for checking if a timed automaton meets a specification that is given as an event-clock automaton.

1 Introduction

Finite automata are instrumental for the modeling and analysis of many phenomena within computer science. In particular, automata theory plays an important role in the verification of concurrent finite-state systems [12, 18]. In the trace model for concurrent computation, a system is identified with its behaviors. Assuming that a behavior is represented as a sequence of states or events, the set of possible behaviors of a system is a formal language, and the system can be modeled as an automaton that generates the language (a complex system is modeled as the product of automata that model the component systems). Since the admissible behaviors of the system also constitute a formal language, the requirements specification can be given by another automaton (the adequacy of automata as a specification formalism is justified by the fact that competing formalisms such as linear temporal logic are no more expressive). The verification problem of checking that a system meets its specification, then, reduces to testing language inclusion between two automata. The decision procedure for language inclusion typically involves the complementation of the specification automaton, which in turn relies upon determinization [11, 17].

*An abbreviated version of this paper appeared in the *Proceedings of the Sixth Annual Conference on Computer-aided Verification*, Lecture Notes in Computer Science 818, Springer-Verlag, 1994, pp. 1–13.

†AT&T Bell Laboratories, Murray Hill, NJ. Email: alur@research.att.com.

‡Department of Computer Science, Cornell University, Ithaca, NY. Email: fix@cs.cornell.edu. Supported in part by the Office of Naval Research under contract N00014-91-J-1219, the National Science Foundation under grant CCR-8701103, and by DARPA/NSF under grant CCR-9014363.

§Department of Computer Science, Cornell University, Ithaca, NY. Email: tah@cs.cornell.edu. Supported in part by the National Science Foundation under grant CCR-9200794, by the United States Air Force Office of Scientific Research under contract F49620-93-1-0056, and by the Defense Advanced Research Projects Agency under grant NAG2-892.

To capture the behavior of a real-time system, the model of computation needs to be augmented with a notion of time. For this purpose, *timed automata* [3] provide a simple, and yet powerful, way of annotating state-transition graphs with timing constraints, using finitely many real-valued variables called *clocks*. A timed automaton, then, accepts *timed words*—strings in which each symbol is paired with a real-valued time-stamp. The theory of timed automata allows the automatic verification of certain real-time requirements of finite-state systems [1, 3, 4, 10], and the solution of certain delay problems [2, 7]. Solutions based on this theory have been implemented in verification tools including COSPAN [6] and KRONOS [10]. However, the general verification problem (i.e., language inclusion) is undecidable for timed automata [3]. This is because, unlike in the untimed case, the nondeterministic variety of timed automata is strictly more expressive than the deterministic variety. The notion of nondeterminism allowed by timed automata, therefore, seems too permissive, and we hesitate to accept timed automata as the canonical model for finite-state real-time computation [5].

In this paper, we obtain a determinizable class of timed automata 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 alphabet symbols typically represent events). 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 ECA of event-clock automata is closed under all boolean operations (timed automata are not closed under complement), and the language inclusion problem is decidable (in PSPACE) for event-clock automata.

The class of event-clock automata is sufficiently expressive to model real-time systems, and to specify real-time requirements. For instance, the typical real-time properties such as "every request is followed by a response within 3 seconds," or "every two consecutive requests are separated by at least 5 seconds," are expressible using event-clock automata. In fact, we argue that automata that contain only event-recording clocks (*event-recording automata*) are a suitable abstract model for real-time systems by proving that event-recording automata are as powerful as another popular model for real-time computation, *timed transition systems* [9]. A timed transition system associates with each transition a lower bound and an upper bound on the time that the transition may be enabled without being taken (many related real-time formalisms also use lower and upper time bounds to express timing constraints [15, 16]). A run of a timed transition system, then, is again a timed word—a sequence of time-stamped state changes. We construct, for a given timed transition system T with a finite set of states, an event-recording automaton that accepts precisely the runs of T. This result leads to a PSPACE algorithm for checking the equivalence of two finite timed transition systems.

The remaining paper is organized as follows. Section 2 defines event-clock automata, while Section 3 shows equivalence between nondeterministic and deterministic classes. Section 4 studies closure properties and decision problems. The final section shows how to use event-clock automata to obtain decision procedures for timed transition systems.

2 Event-clock Automata

Timed words and timed languages

We study formal languages of timed words.[1] A *timed word* $\overline{w}$ over an alphabet Σ is a finite sequence $(a_0, t_0)(a_1, t_1)\ldots(a_n, t_n)$ of symbols $a_i \in \Sigma$ that are paired with nonnegative real numbers $t_i \in \mathrm{R}$ such that the sequence $\overline{t} = t_0 t_1 \ldots t_n$ of time-stamps is nondecreasing (i.e., $t_i \leq t_{i+1}$ for all $0 \leq i < n$). Sometimes we denote the timed word $\overline{w}$ by the pair $(\overline{a}, \overline{t})$. A *timed language* over the alphabet Σ is a set of timed words over Σ. The boolean operations of union, intersection, and complement of timed languages are defined as usual. Given a timed language $\mathcal{L}$ over the alphabet Σ, the projection $Untime(\mathcal{L})$ is obtained by discarding the time-stamps: $Untime(\mathcal{L}) \subseteq \Sigma^*$ consists of all strings $\overline{a}$ for which there exists a sequence $\overline{t}$ of time-stamps such that $(\overline{a}, \overline{t}) \in \mathcal{L}$.

Automata with clocks

Timed automata are finite-state machines that are constrained with timing requirements so that they accept (or generate) timed words (and thus define timed languages); they were proposed in [3] as an abstract model for finite-state real-time systems. A timed automaton operates with finite control—a finite set of locations and a finite set of real-valued variables called *clocks*. Each edge between locations specifies a set of clocks to be reset (i.e., restarted). The value of a clock always records the amount of time that has elapsed since the last time the clock was reset: if the clock z is reset while reading the i-th symbol of a timed input word $(\overline{a}, \overline{t})$, then the value of z while reading the j-th symbol, for $j > i$, is $t_j - t_i$ (assuming that the clock z is not reset at any position between i and j). The edges of the automaton put certain arithmetic constraints on the clock values; the automaton control may proceed along an edge only when the values of the clocks satisfy the corresponding constraints.

Each clock of a timed automaton, therefore, is a real-valued variable that records the time difference between the current input symbol and a previous input symbol, namely, the input symbol on which the clock was last reset. This association between clocks and input symbols is determined dynamically by the behavior of the automaton. An event-clock automaton, by contrast, employs clocks that have a tight, predefined, association with certain symbols of the input word. Suppose that we model a real-time system so that the alphabet symbols represent events of the system. In most cases, it will suffice to know, for each event, the time that has elapsed since the last occurrence of the event. For example, to model a delay of 1 to 2 seconds between the input and output events of a device, it suffices to use a clock z that records the time that has elapsed since the last input event, and require the constraint $1 < z < 2$ when the output event occurs. This observation leads us to the definition of clocks that have a fixed association with input symbols and cannot be reset arbitrarily.

Event-recording and event-predicting clocks

Let Σ be a finite alphabet. For every symbol $a \in \Sigma$, we write x_a to denote the *event-recording clock* of a. Given a timed word $\overline{w} = (a_0, t_0)(a_1, t_1)\ldots(a_n, t_n)$, the value of the clock x_a at the j-th position of $\overline{w}$ is $t_j - t_i$, where i is the largest position preceding j such that a_i equals a. If no occurrence of a precedes the j-th position of $\overline{w}$, then the value of the clock x_a is "undefined," denoted by $\perp$. We write $\mathrm{R}_\perp = \mathrm{R} \cup \{\perp\}$ for the set of nonnegative real numbers together with the

[1] For the clarity of exposition, we limit ourselves to finite words. Our results can be extended to the framework of ω-languages.

special value $\perp$. Formally, we define for all $0 \leq j \leq n$,

$$\lambda(\overline{w}, j)(x_a) \;=\; \begin{cases} t_j - t_i & \text{if there exists } i \text{ such that } 0 \leq i < j \text{ and } a_i = a \\ & \quad\text{and for all } k \text{ with } i < k < j,\ a_k \neq a, \\ \perp & \text{if } a_k \neq a \text{ for all } k \text{ with } 0 \leq k < j. \end{cases}$$

That is, the event-recording clock x_a behaves exactly like an automaton clock that is reset every time the automaton encounters the input symbol a. The value of x_a, therefore, is determined by the input word, not by the automaton. Auxiliary variables that record the times of last occurrences of events have been used extensively in real-time reasoning, for example, in the context of model-checking for timed Petri nets [20], and in assertional proof methods [13, 16].

Event-recording clocks provide timing information about events in the past. The dual notion of event-predicting clocks provides timing information about future events. For every symbol $a \in \Sigma$, we write y_a to denote the *event-predicting clock* of a. At each position of the timed word $\overline{w}$, the value of the clock y_a indicates the time of the next occurrence of a relative to the time of the current input symbol; the special value $\perp$ indicates the absence of a future occurrence of a. Formally, we define for all $0 \leq j \leq n$,

$$\lambda(\overline{w}, j)(y_a) \;=\; \begin{cases} t_i - t_j & \text{if there exists } i \text{ such that } j < i \leq n \text{ and } a_i = a \\ & \quad\text{and for all } k \text{ with } j < k < i,\ a_k \neq a, \\ \perp & \text{if } a_k \neq a \text{ for all } k \text{ with } j < k \leq n. \end{cases}$$

The event-predicting clock y_a can be viewed as an automaton clock that is reset, every time the automaton encounters the input symbol a, to a nondeterministic negative starting value, and checked for 0 at the subsequent occurrence of a.

We write C_Σ for the set $\{x_a, y_a \mid a \in \Sigma\}$ of event-recording and event-predicting clocks. For each position j of a timed word $\overline{w}$, the *clock-valuation function* $\lambda(\overline{w}, j)$, then, is a mapping from C_Σ to $\mathrm{R}_\perp$. The clock constraints compare clock values to rational constants or to the special value $\perp$. Let $\mathrm{Q}_\perp$ denote the set of nonnegative rational numbers together with $\perp$. Formally, a *clock constraint* over the set C of clocks is a boolean combination of atomic formulas of the form $z \leq c$ and $z \geq c$, where $z \in C$ and $c \in \mathrm{Q}_\perp$. The clock constraints over C are interpreted with respect to clock-valuation functions from C to $\mathrm{R}_\perp$: the atom $\perp = \perp$ evaluates to true, and all other comparisons that involve $\perp$ (e.g., $\perp \geq 3$) evaluate to false. For a clock-valuation function γ and a clock constraint φ, we write $\gamma \models \varphi$ to denote that according to γ the constraint φ evaluates to true.

Syntax and semantics of event-clock automata

An event-clock automaton is a (nondeterministic) finite-state machine whose edges are annotated both with input symbols and with clock constraints over event-recording and event-predicting clocks. Formally, an *event-clock automaton* A consists of

- a finite nonempty input alphabet Σ,
- a finite nonempty set L of locations,
- a set $L_0 \subseteq L$ of start locations,
- a set $L_f \subseteq L$ of accepting locations, and
- a finite set E of edges. Each edge is a quadruple $(\ell, \ell', a, \varphi)$ with a source location $\ell \in L$, a target location $\ell' \in L$, an input symbol $a \in \Sigma$, and a clock constraint φ over the clocks C_Σ.

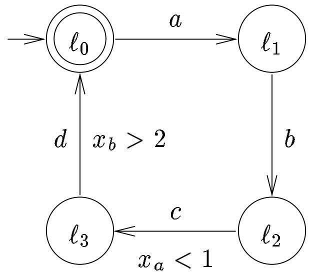

Figure 1: Event-recording automaton A_1

Now let us consider the behavior of an event-clock automaton over the timed input word $\overline{w} = (a_0, t_0)(a_1, t_1)\ldots(a_n, t_n)$. Starting in one of the start locations and scanning the first input pair (a_0, t_0), the automaton scans the input word from left to right, consuming, at each step, an input symbol together with its time-stamp. In location ℓ scanning the i-th input pair (a_i, t_i), the automaton may proceed to location ℓ' and the $(i+1)$-st input pair iff there is an edge $(\ell, \ell', a, \varphi)$ such that a equals the current input symbol a_i and $\lambda(\overline{w}, i)$ satisfies the clock constraint φ. Formally, a *computation* of the event-clock automaton A over the timed input word $\overline{w}$ is a finite sequence

$$\ell_0 \xrightarrow{e_0} \ell_1 \xrightarrow{e_1} \ell_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} \ell_n \xrightarrow{e_n} \ell_{n+1}$$

of locations $\ell_i \in L$ and edges $e_i = (\ell_i, \ell_{i+1}, a_i, \varphi_i) \in E$ such that $\ell_0 \in L_0$ and for all $0 \leq i \leq n$, $\lambda(\overline{w}, i) \models \varphi_i$; the computation is *accepting* if $\ell_{n+1} \in L_f$. The timed language $\mathcal{L}(A)$ defined by the event-clock automaton A, then, consists of all timed words $\overline{w}$ such that A has an accepting computation over $\overline{w}$. We write ECA for the class of timed languages that are definable by event-clock automata.

The event-clock automaton A is an *event-recording automaton* if all clock constraints of A contain only event-recording clocks; A is an *event-predicting automaton* if the clock constraints of A contain only event-predicting clocks. The class of timed languages that can be defined by these two restricted types of event-clock automata are denoted ERA and EPA, respectively.

Examples of event-clock automata

The event-clock automaton A_1 of Figure 1 uses two event-recording clocks, x_a and x_b. The location ℓ_0 is the start location of A_1, and also the sole accepting location. The clock constraint $x_a < 1$ that is associated with the edge from ℓ_2 to ℓ_3 ensures that c occurs within 1 time unit of the preceding a. A similar mechanism of checking the value of x_b while reading d ensures that the time difference between b and the subsequent d is always greater than 2. Thus, the timed language $\mathcal{L}(A_1)$ defined by A_1 consists of all timed words of the form $((abcd)^m, \overline{t})$ such that $m \geq 0$ and for all $0 \leq j < m$, $t_{4j+2} < t_{4j} + 1$ and $t_{4j+3} > t_{4j+1} + 2$. Note that the timed language $\mathcal{L}(A_1)$ can also be defined using event-predicting clocks: require $y_c < 1$ while reading a, and $y_d > 2$ while reading b.

The duality of the two types of clocks is further illustrated by the automata of Figure 2. The event-recording automaton A_2 accepts all timed words of the form $(ab^*b, \overline{t})$ such that the time difference between the two extreme symbols is 1, which is enforced by the event-recording clock x_a. Later we prove that there is no event-predicting automaton that defines the timed language $\mathcal{L}(A_2)$. The event-predicting automaton A_3, on the other hand, accepts all timed words of the form $(aa^*b, \overline{t})$

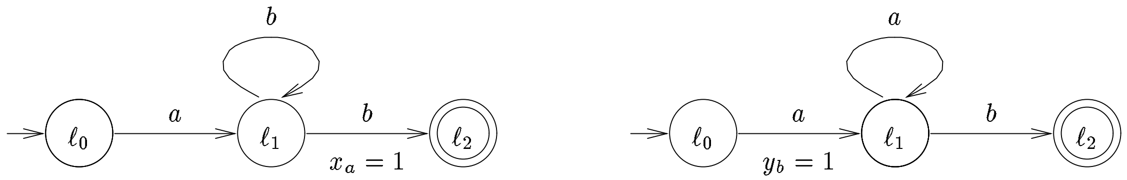

Figure 2: Event-recording automaton A_2 and event-predicting automaton A_3

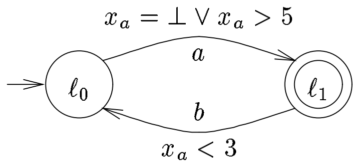

Figure 3: Event-recording automaton A_4

such that the time difference between the two extreme symbols is 1; for this purpose, the event-predicting clock y_b is used to predict the time of the first b. There is no event-recording automaton that defines $\mathcal{L}(A_3)$.

The automaton A_4 of Figure 3 expresses the requirement that every request a is followed by a response b within 3 seconds, and two requests are separated by at least 5 seconds. Examples such the railroad-gate controller and timing-based mutual-exclusion algorithms that appear in the literature on finite-state real-time verification (see, for instance, [3, 10, 6]) can all be formulated using event-clock automata.

3 Deterministic Event-clock Automata

A finite-state machine (with a single start location) is deterministic iff all input symbols that label edges with the same source location are pairwise distinct. We consider for event-clock automata the notion of determinism that was proposed for timed automata in [3]. The event-clock automaton $A = (\Sigma, L, L_0, L_f, E)$ is *deterministic* if

1. A has a single start location (i.e., $|L_0| = 1$), and
2. two edges with the same source location and the same input symbol have mutually exclusive clock constraints; that is, if $(\ell, \ell', a, \varphi_1) \in E$ and $(\ell, \ell'', a, \varphi_2) \in E$ then for all clock-valuation functions γ, if $\gamma \models \varphi_1$ then $\gamma \not\models \varphi_2$.

The determinism condition ensures that at each step during a computation, the choice of the next edge is uniquely determined by the current location of the automaton, the input word, and the current position of the automaton along the input word. It is easy to check that every deterministic event-clock automaton has at most one computation over any given timed input word.

Of our examples from the previous section, the event-clock automata A_1, A_3, and A_4 are deterministic. While the automaton A_2 is nondeterministic, it can be determinized without changing its language, by adding the clock constraint $x_a < 1$ to the self-loop at location ℓ_1.

In the theory of finite-state machines, it is well-known that every nondeterministic automaton can be determinized; that is, the deterministic and nondeterministic varieties of finite-state machines define the same class of languages (the regular languages). In the case of timed automata, however, the nondeterministic variety is strictly more expressive than its deterministic counterpart [3]. We now show that the event-clock automata form a determinizable subclass of the timed automata.

The determinization follows the standard subset construction. Let $A = (\Sigma, L, L_0, L_f, E)$ be the given event-clock automaton. The determinized automaton $Det(A)$ over the same alphabet Σ has the following components:

- The locations of $Det(A)$ are the subsets of L.
- The only start location is L_0.
- A location $L' \subseteq L$ is an accepting location iff $L_f \cap L'$ is nonempty.
- Consider a location $L' \subseteq L$ of $Det(A)$ and an input symbol $a \in \Sigma$. Let $E' \subseteq E$ be the set of all a-labeled edges of A whose source locations are in L'. Then, for every subset E'' of E', there is an edge from L' to L'' labeled with the input symbol a and the clock constraint φ such that
  - L'' contains precisely the target locations of the edges in E'', and
  - φ is the conjunction of all clock constraints of edges in E'' and all negated clock constraints of edges in $(E' - E'')$.

It is easy to check the following properties of $Det(A)$:

1. $\mathcal{L}(A) = \mathcal{L}(Det(A))$.
2. Given a location L' of $Det(A)$, an input symbol a, and a clock-valuation γ, there is precisely one edge (L', L'', a, φ) such that $\gamma \models \varphi$.
3. $Det(A)$ is deterministic.
4. $Det(A)$ is an event-recording (event-predicting) automaton if A is an event-recording (event-predicting) automaton.

Theorem 1 (Determinization) *For every event-clock (event-recording; event-predicting) automaton A, there is a deterministic event-clock (event-recording; event-predicting) automaton that defines $\mathcal{L}(A)$.*

Notice that the determinization of an event-clock automaton causes an exponential blow-up in the number of locations, but changes neither the number of clocks nor the constants that occur in clock constraints.

The key for the determinization of event-clock automata is the property that at each step during a computation, all clock values are determined solely by the input word. We therefore obtain determinizable superclasses of event-clock automata if we add more clocks that do not violate this property. For example, for each input symbol a and each natural number i, we could employ a clock z_a^i that records the time since the i-th occurrence of a, and a clock x_a^i that records the time since the i-th-to-last occurrence of a (i.e., $x_a = x_a^1$). Or, more ambitiously, we may want to use for each linear temporal formula ψ a *formula-recording clock* x_ψ that measures the time since the last position of the input word at which ψ was true, and a *formula-predicting clock* y_ψ that measures the time until the next position at which ψ will be true.

4 Properties of Event-Clock Automata

Event-clock automata as labeled transition systems

We now consider an alternative semantics for event-clock automata, using labeled transition systems. Let $A = (\Sigma, L, L_0, L_f, E)$ be an event-clock automaton. We define an infinite-state labeled transition system T_A to capture the behavior of A over timed words. The transition system T_A has the following components:

- A *state* of T_A is a pair (ℓ, γ) that consists of a location $\ell \in L$ and a clock-valuation function γ from C_Σ to $\mathrm{R}_\perp$, which determines the values of all clocks. The state-space of T_A is denoted by S_A.
- The set $S_A^0 \subseteq S_A$ of initial states consists of states (ℓ, γ) such that $\ell \in L_0$ and $\gamma(x_a) = \perp$ for all input symbols $a \in \Sigma$.
- The set $S_A^f \subseteq S_A$ of final states consists of states (ℓ, γ) such that $\ell \in L_f$ and $\gamma(y_a) = \perp$ for all $a \in \Sigma$.
- For two states $s, s' \in S_A$, an input symbol $a \in \Sigma$, and a real-valued time delay $\delta \in \mathrm{R}$, let $s \xrightarrow{a} s'$ if the automaton A may proceed from the state s to the state s' by reading the input symbol a, and let $s \xrightarrow{\delta} s'$ if A may proceed from s to s' by letting time δ pass. Formally,
  - $(\ell, \gamma) \xrightarrow{a} (\ell', \gamma')$ iff there is a clock-valuation γ'' and an edge $(\ell, \ell', a, \varphi) \in E$ such that
    - $\gamma = \gamma''[y_a := 0]$ (i.e., γ agrees with γ'' on all clocks except y_a, which in γ evaluates to 0),
    - $\gamma' = \gamma''[x_a := 0]$, and
    - $\gamma'' \models \varphi$.
  - $(\ell, \gamma) \xrightarrow{\delta} (\ell', \gamma')$ iff $\ell = \ell'$ and for all input symbols $b \in \Sigma$,
    - if $\gamma(x_b) = \perp$ then $\gamma'(x_b) = \perp$, else $\gamma'(x_b) = \gamma(x_b) + \delta$; and
    - if $\gamma'(y_b) = \perp$ then $\gamma(y_b) = \perp$, else $\gamma(y_b) = \gamma'(y_b) + \delta$.

We inductively extend the labeled transition relation to timed words:

- $s \xrightarrow{(a_0, t_0)} s'$ if there is a state $s'' \in S_A$ such that $s \xrightarrow{t_0} s''$ and $s'' \xrightarrow{a_0} s'$;
- if $\overline{w} = (a_0, t_0) \ldots (a_n, t_n)$ and $\overline{w}' = \overline{w}(a_{n+1}, t_{n+1})$, then $s \xrightarrow{\overline{w}'} s'$ if there is a state s'' such that
$$s \xrightarrow{\overline{w}} s'' \text{ and } s'' \xrightarrow{(a_{n+1}, t_{n+1} - t_n)} s'.$$

The following lemma states the correctness of the labeled-transition-system semantics for event-clock automata.

Lemma 1 *The event-clock automaton A accepts the timed word $\overline{w}$ iff $s \xrightarrow{\overline{w}} s'$ for some initial state s and some final state s' of T_A.*

The region construction

The analysis of timed automata builds on the so-called region construction, which transforms a timed automaton into an untimed finite-state machine [1, 3]. Here we apply the region construction to event-clock automata.

Consider an event-clock automaton A and the corresponding transition system T_A. An equivalence relation $\cong$ on the state-space S_A is said to be a *time-abstract bisimulation* iff for all states $s_1, s_2 \in S_A$, if $s_1 \cong s_2$ then

1. if $s_1 \xrightarrow{a} s_1'$ for some input symbol a then there exists a state s_2' with $s_2 \xrightarrow{a} s_2'$ and $s_1' \cong s_2'$,
2. if $s_1 \xrightarrow{\delta} s_1'$ for some delay δ then there exists a state s_2' and a delay δ' with $s_2 \xrightarrow{\delta'} s_2'$ and $s_1' \cong s_2'$.

For a time-abstract bisimulation $\cong$, the *region automaton* $Reg_{\cong}(A)$ is an (untimed) automaton over the input alphabet Σ. A $\cong$*-region* is an equivalence class of the relation $\cong$. The locations of $Reg_{\cong}(A)$ are the $\cong$-regions of A. A $\cong$-region is starting if it contains an initial state of T_A, and accepting if it contains a final state of T_A. There is an edge from the $\cong$-region ρ to the $\cong$-region ρ' labeled with the input symbol a if there are two states $s \in \rho$ and $s' \in \rho'$, and a time delay $\delta \in \mathrm{R}$, such that $s \xrightarrow{(a,\delta)} s'$. From Lemma 1 and the definition of time-abstract bisimulation, it follows that the region automaton $Reg_{\cong}(A)$ defines the untimed language $Untime(\mathcal{L}(A))$.

It follows that the emptiness problem for event-clock automata can be solved if there is an (effectively computable) time-abstract bisimulation with only finitely many regions. The region-equivalence relation $\cong_A$ is one such relation. We assume that all clock constraints of A contain only integer constants (otherwise, all constants need to be multiplied by the least common multiple of the denominators of all rational numbers that appear in the clock constraints of A). Let c be the largest integer constant that appears in a clock constraint of A. Two clock-valuation functions γ and γ' from C_Σ to $\mathrm{R}_\perp$ are *region-equivalent*, written $\gamma \cong_A \gamma'$, iff all the following conditions are satisfied

1. γ and γ' agree on which clocks have the undefined value: for all $z \in C_\Sigma$, $\gamma(z) = \perp$ iff $\gamma'(z) = \perp$.
2. γ and γ' agree on the integral parts of all defined clock values that are at most c: for all $z \in C_\Sigma$, if $\gamma(z) \leq c$ or $\gamma'(z) \leq c$ then $\lfloor \gamma(z) \rfloor = \lfloor \gamma'(z) \rfloor$.
3. γ and γ' agree on the ordering of the fractional parts of all defined clock values that are at most c. For an event-recording clock x_a, let $\langle \gamma(x_a) \rangle$ be $\gamma(x_a) - \lfloor \gamma(x_a) \rfloor$; for an event-predicting clock y_a, let $\langle \gamma(y_a) \rangle$ be $\lceil \gamma(y_a) \rceil - \gamma(y_a)$. For all $z, z' \in C_\Sigma$ with $\gamma(z) \leq c$ and $\gamma(z') \leq c$,
   - $\langle \gamma(z) \rangle = 0$ iff $\langle \gamma'(z) \rangle = 0$, and
   - $\langle \gamma(z) \rangle \leq \langle \gamma(z') \rangle$ iff $\langle \gamma'(z) \rangle \leq \langle \gamma'(z') \rangle$.

Two states $(\ell, \gamma), (\ell', \gamma') \in S_A$ are *region-equivalent* if $\ell = \ell'$ and $\gamma \cong_A \gamma'$.

Lemma 2 *The region-equivalence relation $\cong_A$ is a time-abstract bisimulation.*

Henceforth, a region of A means $\cong_A$-region, and the region automaton of A means the automaton $Reg_{\cong_A}(A)$. The number of regions of A is finite.

Lemma 3 *The number of equivalence classes of the region-equivalence relation $\cong_A$ is $n \cdot 2^{O(m \log cm)}$, where n is the number of locations of A, m is the number of clocks (i.e. the size of the alphabet), and c is the largest constant appearing in a clock constraint of A.*

This implies that the region automaton is a finite automaton.

Theorem 2 (Untiming) *For every event-clock automaton A, the untimed language $Untime(\mathcal{L}(A))$ is regular.*

Closure properties and decision problems

While the class of timed automata is not closed under complement, and the language inclusion (verification) problem for timed automata is undecidable, the subclass of event-clock automata is well-behaved.

Theorem 3 (Closure properties) *Each of the classes* ECA, ERA, *and* EPA *of timed languages are closed under union, intersection, and complement.*

Proof. Closure under union is trivial, because event-clock automata admit multiple start locations.

Closure under intersection is also straightforward, because the standard automata-theoretic product construction $A_1 \times A_2$ for two given event-clock (event-recording; event-predicting) automata A_1 and A_2 yields an event-clock (event-recording; event-predicting) automaton. Each location of $A_1 \times A_2$ consists of a location of A_1 and a location of A_2, and each a-edge e of $A_1 \times A_2$ corresponds to both an a-edge e_1 of A_1 and an a-edge of A_2 (the clock constraint of e is the conjunction of the clock constraints of e_1 and e_2).

Closure under complement relies on the determinization construction: given an event-clock (event-recording; event-predicting) automaton A, the event-clock (event-recording; event-predicting) automaton $\neg Det(A)$ that results from complementing the acceptance condition of $Det(A)$ (interchange the accepting and the nonaccepting states of $Det(A)$) defines the complement of the timed language $\mathcal{L}(A)$. ■

Unlike (nondeterministic) timed automata, however, event-clock automata are not closed under hiding and renaming of input symbols. Consider the timed language $\mathcal{L}$ over a unary alphabet that contains all the timed words $\overline{w} = (\overline{a}, \overline{t})$ in which no two symbols occur with time difference 1 (i.e., $t_j - t_i \neq 1$ for all pairs of positions i and j of $\overline{w}$). The timed language $\mathcal{L}$ cannot be defined by a timed automaton [3], and hence, by an event-clock automaton. This fact can be used to prove nonclosure properties. For instance, consider the timed language $\mathcal{L}'$ containing timed words $(a^*ba^*ba^*, \overline{t})$ such that the time difference of the two b-symbols is 1. The timed language $\mathcal{L}'$ is definable by an event-recording or an event-predicting automaton, and thus, is in ERA $\cap$ EPA. If we rename the input symbol b to a, the resulting timed language contains all the timed words $\overline{w} = (\overline{a}, \overline{t})$ over a unary alphabet in which some two symbols occur with time difference 1 (i.e., $t_j - t_i = 1$ for two positions i and j of $\overline{w}$), precisely the complement of the timed language $\mathcal{L}$. Since the classes are closed under complementation, it follows that neither ERA nor EPA is closed under renaming. The nondefinability of $\mathcal{L}$ can also be used to show nonclosure under hiding.

The determinization, closure properties, and region construction can be used to solve decision problems for event-clock automata. To check if the timed language of an event-clock automaton A is empty, we construct the region automaton $Reg(A)$ and check if the untimed language of $Reg(A)$ is empty. Since the number of regions is exponential, various heuristics have been proposed to solve emptiness (and other reachability problems) more efficiently. For instance, it is possible to construct the time-abstract bisimulation with minimal number of regions [19] using minimization algorithms, or to incorporate the timing constraints of the automaton in an iterative way generating successive approximations to the region automaton [2]. The reachability problems can also be solved by a symbolic fixpoint computation procedure [8, 10].

To check if the language of the event-clock automaton A_1 is included in the language of the event-clock automaton A_2, we determinize A_2, complement $Det(A_2)$, take the product with A_1, and check if the language of the resulting event-clock automaton is empty by constructing the corresponding region automaton.

Theorem 4 (Language inclusion) *The problem of checking if $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ for two event-clock automata A_1 and A_2 is* PSPACE*-complete.*

Proof. Consider two event-clock automata A_1 and A_2 such that each automaton has at most n locations, and let m be the size of the alphabet. The first step involves multiplying all constants in the clock constraints by the least common multiple of the denominators so that the clock constraints have only integer constants. Let c be the largest integer constant appearing in the clock constraints after this normalization step. The length of c (i.e. the number of bits) is at most quadratic in the length of the encoding of the original constraints. Let $\neg Det(A_2)$ be the complement of $Det(A_2)$. The automaton $\neg Det(A_2)$ has 2^n locations, and the constants in the clock constraints of $\neg Det(A_2)$ are bounded by c. Let A_3 be the product of A_1 and $\neg Det(A_2)$. The automaton A_3 has $n \cdot 2^n$ locations, and the constants in the clock constraints of A_3 are also bounded by c. The size of the alphabet (and hence the number of clocks) for A_3 is m. To test if the language of A_3 is empty, we need to construct the corresponding region automaton $Reg(A_3)$. By Lemma 3, the region automaton $Reg(A_3)$ has $n \cdot 2^n \cdot 2^{O(m\ log\ cm)}$ regions, that is, singly exponential in the length of the input automata. Checking emptiness corresponds to searching for accepting paths in this exponential-sized automaton. Since the rules defining the edges of $Reg(A_3)$ are straightforward, it follows that the emptiness can be checked in PSPACE.

On the other hand, the problem of checking if the language of a given event-recording (or event-predicting) automaton is empty is PSPACE-hard. The proof is same as the corresponding hardness proof for timed automata [3]. ■

Relationship among classes of timed automata

We briefly review the definition of a timed automaton [3]. A (nondeterministic) *timed automaton* A consists of a finite input alphabet Σ, a finite set L of locations, a set $L_0 \subseteq L$ of start locations, a set $L_f \subseteq L$ of accepting locations, a finite set C of clocks, and a finite set E of edges. Each edge e consists of a source location ℓ, a target location ℓ', an input symbol a, a clock constraint φ over C, and a *reset condition* $r \subseteq C$ that specifies the clocks that are reset to 0 when the edge e is traversed. A clock-valuation for the timed automaton A is a function from the clocks C to the reals $\mathrm{R}_\perp$. A computation of the timed automaton A over the timed input word $\overline{w}$ is a finite sequence

$$(\ell_0, \gamma_0) \xrightarrow{e_0} (\ell_1, \gamma_1) \xrightarrow{e_1} \cdots \xrightarrow{e_{n-1}} (\ell_n, \gamma_n) \xrightarrow{e_n} (\ell_{n+1}, \gamma_{n+1})$$

of locations $\ell_i \in L$, clock-valuations γ_i, and edges $e_i = (\ell_i, \ell_{i+1}, a_i, \varphi_i, r_i) \in E$ such that $\ell_0 \in L_0$, $\gamma_0(x) = \perp$ for all $x \in C$, $\gamma_0 \models \varphi_0$, $\gamma_1 = \gamma_0[r_0 := 0]$, and for all $1 \leq i \leq n$, $\gamma_i + (t_i - t_{i-1}) \models \varphi_i$ and $\gamma_{i+1} = (\gamma_i + t_i - t_{i-1})[r_i := 0]$. The computation is *accepting* if $\ell_{n+1} \in L_f$. The timed language $\mathcal{L}(A)$ consists of all timed words $\overline{w}$ such that A has an accepting computation over $\overline{w}$, and we write NTA for the class of timed languages that are definable by timed automata. Figure 4 shows a timed automaton with a single clock x over the unary alphabet. The timed language $\mathcal{L}(A)$ consists of all timed words $(a^n, \overline{t})$ such that $t_j - t_i = 1$ for some $0 \leq i < j \leq n$.

The class NTA is closed under union and intersection, but not under complement. For the automaton A of Figure 4, the complement language is not definable. Testing emptiness of the

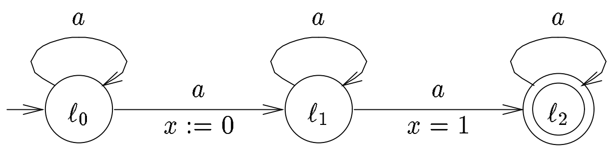

Figure 4: Timed automaton A

language of a timed automaton is PSPACE-complete, while testing inclusion between the languages of two timed automata is undecidable [3].

The definition of determinism for timed automata is the same as for event-clock automata. We write DTA for the class of timed languages that are definable by deterministic timed automata. Since DTA is closed under all boolean operations, DTA is strictly contained in NTA.

Next, consider the translation from event-clock automata to timed automata. Translating event-recording clocks is easy: the event-recording clock x_a is reset on an edge iff the corresponding input symbol is a. This trivial translation preserves determinism. Translation of event-predicting clocks introduces nondeterminism.

Consider an event-predicting automaton $A = (\Sigma, L, L_0, L_f, E)$. A comparison operator $\sim$ is either $\leq$ or $<$ or $\geq$ or $>$. An *atomic* clock-constraint is an atomic formula $y_a \sim c$ or $y_a = \perp$. Assume that each edge of A is labeled with a conjunction of atomic clock constraints (this can be achieved by a straightforward rewriting of negations and disjunctions, and replacing disjunctions by multiple edges). Let Φ_A be the set of atomic clock constraints that appear in the constraints of A. Construct a nondeterministic timed automaton B over the same alphabet Σ as follows:

- A location of B is a pair (ℓ, Ψ) with $\ell \in L$ and $\Psi \subseteq \Phi_A$.
- A location (ℓ, Ψ) is a start location iff $\ell \in L_0$ and Ψ does not contain a constraint of the form $y_a \sim c$.
- A location (ℓ, Ψ) is an accepting location iff $\ell \in L_f$ and Ψ equals $\{y_a = \perp \mid a \in \Sigma\}$.
- For every (atomic) constraint $\psi \in \Phi_A$, the automaton B has a clock z_ψ.
- The automaton B has an edge from (ℓ, Ψ) to (ℓ', Ψ') labeled with the input symbol a, clock constraint φ, and reset condition r iff all the following conditions hold. Intuitively, a prediction $y_b \sim c$ along an edge in A on the time difference to the next occurrence of b is replaced in B by a constraint on the clock $z_{y_b \sim c}$. The clock $z_{y_b \sim c}$ is reset to zero when the prediction is made and its value is checked by the constraint $z_{y_b \sim c} \sim c$ when the next b occurs.
  1. the automaton A has an edge (ℓ, ℓ', a, χ).
  2. the constraint $y_a = \perp$ is not in Ψ.
  3. the constraint φ is the conjunction $\wedge_{(y_a \sim c) \in \Psi} z_{(y_a \sim c)} \sim c$.
  4. for all input symbols $b \neq a$, if a constraint involving y_b appears in Ψ then it appears in Ψ' also.
  5. if an atomic constraint is a conjunct of χ then it appears in Ψ' also.

6. for all input symbols b and for $\sim$ equal to $>$ or $\geq$, a clock $z_{(y_b \sim c)}$ is in the reset set r iff the constraint $y_b \sim c$ is a conjunct of χ.
7. for all input symbols b and for $\sim$ equal to $<$ or $\leq$, a clock $z_{(y_b \sim c)}$ is in the reset set r iff the constraint $y_b \sim c$ is a conjunct of χ and either $b = a$ or the constraint $y_b \sim c$ is not in Ψ.

The automaton B defines the timed language $\mathrm{L}(A)$. The next theorem gives relationships among various classes.

Theorem 5 (Relationship between classes)

(1) ERA $\not\subseteq$ EPA	(2) EPA $\not\subseteq$ ERA	(3) ERA $\cup$ EPA $\subset$ ECA
(4) ECA $\subset$ NTA	(5) ERA $\subset$ DTA	(6) EPA $\not\subseteq$ DTA
(7) DTA $\not\subseteq$ ECA		

Proof. For (1), the language of the event-recording automaton A_2 of Figure 2 is not definable by an event-predicting automaton. Suppose an event-predicting automaton B defines the timed language $\mathcal{L}(A_2)$. Without loss of generality, assume that the clock constraints of B use only integer constants. Consider two timed words $\overline{w}_1 = (a,0)(b,0.5)(b,1)$ and $\overline{w}_2 = (a,0)(b,0.5)(b,0.9)$. The event-predicting automaton B uses constraints over two clocks y_a and y_b. Even though $\lambda(\overline{w}_1,1)(y_b)$ differs from $\lambda(\overline{w}_2,1)(y_b)$, clock-constraints with integer constants cannot detect this difference, and hence, for every clock constraint φ and every position j, $\lambda(\overline{w}_1,j) \models \varphi$ iff $\lambda(\overline{w}_2,j) \models \varphi$. Thus, the automaton B accepts $\overline{w}_1$ iff it accepts $\overline{w}_2$. But, the automaton A_2 accepts $\overline{w}_1$ and rejects $\overline{w}_2$.

For (2), the language of the event-predicting automaton A_3 of Figure 2 cannot be defined by an event-recording automaton. The proof is similar to the case (1).

For (3), consider the union of the automata A_2 and A_3. The resulting automaton is an event-clock automaton. A proof as in case (1) shows that the timed language $\mathcal{L}(A_2) \cup \mathcal{L}(A_3)$ is neither in ERA nor in EPA. This shows that the inclusion ERA $\cup$ EPA $\subseteq$ ECA is strict.

The translation from event-clock automata to timed automata proves the inclusions (4) and (5). Inclusion (4) is strict, because ECA is closed under complement while NTA is not. Inclusion (5) is strict because of (7).

For (6), the timed language $\{(a^n b, t_0 \ldots t_n) \mid \exists 0 \leq i < n. t_n - t_i = 1\}$ is in EPA (the event-predicting automaton simply requires that the clock-constraint $y_b = 1$ is satisfied at one of the symbols in the initial string of a-s) but not in DTA. The proof is by contradiction. Suppose there is a deterministic timed automaton B that accepts this language. Suppose B uses only integer constants, and k clocks. Consider the timed word $\overline{w} = (a^{k+1}, t_0 \ldots t_k)$ with $0 < t_0 < t_1 < \cdots < t_{k-1} < t_k < 1$. Since B is deterministic, there is at most one computation that reads the word $\overline{w}$. Since B has at most k clocks, there is at least one index $0 \leq j \leq k$ such that no clock of B is reset along this computation while reading the input (a, t_j). Let $t' \neq t_j$ be such that $t_{j-1} < t' < t_{j+1}$ (assume $t_{j-1} = 0$ if $j = 0$, and $t_{j+1} = 1$ if $j = k$). Thus, the automaton B "forgets" the time value t_j. Consider two extensions of the timed word $\overline{w}$: let $\overline{w}_1$ be $\overline{w}(a, t_j + 1)$ and $\overline{w}_2$ be $\overline{w}(a, t' + 1)$. The clocks of B satisfy the same set of constraints while reading the new input symbol in both cases. Thus, the automaton B either accepts both $\overline{w}_1$ and $\overline{w}_2$ or rejects both, but $\overline{w}_1$ is in the given timed language, and $\overline{w}_2$ is not.

For (7), the timed language $\{(aaa, t_0 t_1 t_2) \mid t_2 - t_0 = 1\}$ is in DTA (the deterministic automaton with one clock x simply resets x while reading the first a, and checks the clock constraint $x = 1$ while reading the third symbol) but not in ECA (the proof is similar to the case (1), an event-clock automaton either accepts both $(a,0)(a,0.5)(a,1)$ and $(a,0)(a,0.5)(a,0.9)$ or rejects both). ■

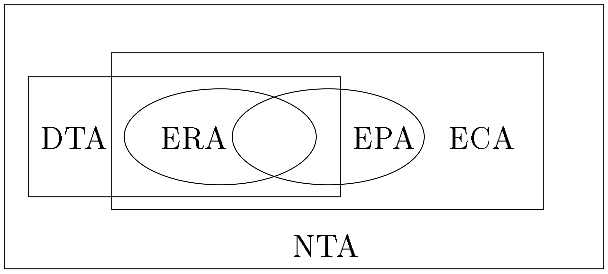

Figure 5: Relationship among classes of timed automata

The relationship among various classes of timed automata is shown in Figure 5. In [5], we defined another subclass of NTA that is closed under all boolean operations, namely, the class 2DTA of timed languages that are definable by deterministic two-way automata that can read the input word a bounded number of times. While ECA is easily seen to be contained in 2DTA, and while there are obvious similarities between event-predicting clocks and the two-way reading of the timed input word, the exact relationship between event-clock automata and deterministic two-way automata remains to be studied. However, because they admit nondeterminism, event-clock automata are perhaps more suited for specification than deterministic two-way automata.

5 Timed Transition Systems as Event-clock Automata

Timed transition systems

A *transition system* T consists of a set S of states, a set $S_0 \subseteq S$ of initial states, and a finite set $\mathcal{T}$ of transitions. Each transition $\tau \in \mathcal{T}$ is a function from S to 2^S; for each state $s \in S$, the set $\tau(s)$ gives the possible τ-successors of s. The transition system T is *finite* if the set S of states is finite. A *run* $\overline{s}$ of the transition system T is a finite sequence $s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$ of states such that $s_0 \in S_0$ and for all $0 \leq i < n$, there exists a transition $\tau_i \in \mathcal{T}$ such that $s_{i+1} \in \tau_i(s_i)$. The transition τ is *enabled* at the i-th step of the run $\overline{s}$ if $\tau(s_i)$ is nonempty, and τ is *taken* at the i-th step if $s_i \in \tau(s_{i-1})$ (note that multiple transitions may be taken at the same step). A variety of programming systems, such as message-passing systems and shared-memory systems, can be given a transition-system semantics [14].

The model of transition systems is extended to timed transition systems so that it is possible to express real-time constraints on the transitions [9]. A *timed transition system* T consists of a transition system $(S, S_0, \mathcal{T})$ and two functions l and u from $\mathcal{T}$ to R that associate with each transition $\tau \in \mathcal{T}$ a *lower bound* $l(\tau)$ and an upper bound $u(\tau)$. Informally, the transition τ must be enabled continuously for at least $l(\tau)$ time units before it can be taken, and τ must not be enabled continuously for more than $u(\tau)$ time units without being taken. Formally, we associate a real-valued time-stamp with each state change along a run: t_0 is the initial time, and the transition system proceeds from the state s_i to the state s_{i+1} at time t_{i+1}. A *timed run* $\overline{r}$ of the timed transition system T is a finite sequence

$$\xrightarrow{t_0} s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} s_n$$

of states $s_i \in S$ and nondecreasing time-stamps $t_i \in \mathrm{R}$ such that $\overline{s}$ is a run of the underlying transition system and

1. *Upper Bound*: if τ is enabled at all steps k for $i \leq k < j$, and not taken at all steps k for $i < k < j$, then $t_j - t_i \leq u(\tau)$;

2. *Lower Bound*: if τ is taken at the j-th step then there is some step $i < j$ such that $t_j - t_i \geq l(\tau)$ and τ is enabled at all steps k for $i \leq k < j$, and not taken at all steps k for $i < k < j$.

The semantics of the timed transition system T is the set of timed runs of T. Two timed transition systems are *equivalent* if they have the same timed runs.

From timed transition systems to event-recording automata

We now show that the set of timed runs of a finite timed transition system can be defined by an event-recording automaton. For this purpose, we need to switch from the state-based semantics of transition systems to an event-based semantics. With the given timed run $\overline{r}$ with states s_i and time-stamps t_i, we associate the timed word

$$\overline{w}(\overline{r}) \;=\; (\langle \perp, s_0\rangle, t_0)\ (\langle s_0, s_1\rangle, t_1)\ (\langle s_1, s_2\rangle, t_2)\ \ldots\ (\langle s_{n-1}, s_n\rangle, t_n),$$

where $\perp$ is a special symbol not in S (as usual, $S_\perp = S \cup \{\perp\}$). Notice that the timed run $\overline{r}$ and the corresponding timed word $\overline{w}(\overline{r})$ contain the same information: each event (i.e., state change) of $\overline{r}$ is modeled by a pair of states—a source state and a target state. Every finite timed transition system $T = (S, \mathcal{T}, S_0, l, u)$, then, defines a timed language $\mathcal{L}(T)$ over the alphabet $S_\perp \times S$, namely, the set of timed words $\overline{w}(\overline{r})$ that correspond to timed runs $\overline{r}$ of T. It is easy to check that two timed transition systems are equivalent iff they define the same timed language.

Theorem 6 (Timed transition systems) *For every finite timed transition system T, there is an event-recording timed automaton A_T that defines the timed language $\mathcal{L}(T)$.*

Proof. Consider the given finite timed transition system T. Each location of the corresponding event-clock automaton A_T records a state $s \in S$ and, for each transition $\tau \in \mathcal{T}$, a pair of states $\langle \alpha(\tau), \beta(\tau)\rangle \in S_\perp \times S$ such that if τ is enabled in s, then τ has been enabled continuously without being taken since the last state change from $\alpha(\tau)$ to $\beta(\tau)$. In addition, we use a special location ℓ_0 as the sole start location of A_T. Every location is an accepting location.

For every initial state $s_0 \in S_0$, there is an edge from ℓ_0 to $(s_0, \langle \alpha, \beta\rangle)$ labeled with the input symbol $\langle \perp, s_0\rangle$ and the trivial clock constraint *true*, where $\alpha(\tau) = \perp$ and $\beta(\tau) = s_0$ for all transitions $\tau \in \mathcal{T}$. In addition, there is an edge from $(s, \langle \alpha, \beta\rangle)$ to $(s', \langle \alpha', \beta'\rangle)$ labeled with the input symbol $\langle s, s'\rangle$ and the clock constraint φ iff there is a transition $\tau \in \mathcal{T}$ such that $(s, s') \in \tau$, and for all transitions $\tau \in \mathcal{T}$,

1. if τ is enabled in s and $s' \notin \tau(s)$, then $\langle \alpha'(\tau), \beta'(\tau)\rangle = \langle \alpha(\tau), \beta(\tau)\rangle$, else $\langle \alpha'(\tau), \beta'(\tau)\rangle = \langle s, s'\rangle$;

2. if τ is enabled in s, then φ contains the conjunct $x_{\langle \alpha(\tau), \beta(\tau)\rangle} \leq u(\tau)$;

3. if $s' \in \tau(s)$, then φ contains the conjunct $x_{\langle \alpha(\tau), \beta(\tau)\rangle} \geq l(\tau)$. ■

Notice that the event-recording automaton A_T is deterministic, and its size is exponential in the size of the timed transition system T. To check if two timed transition systems T_1 and T_2 are equivalent, we construct the corresponding event-recording automata A_{T_1} and A_{T_2} and check if they define the same timed language.

Theorem 7 (Equivalence of timed transition systems) *The problem of checking if two finite timed transition systems are equivalent is* PSPACE*-complete.*

Proof. Checking whether two nondeterministic finite-state automata accept the same language is PSPACE-hard, and hence, checking if two timed transition systems are equivalent is PSPACE-hard. For the upper bound, consider two finite timed transition systems T_1 and T_2 with n states. Suppose that each transition system has at most k transitions, and the bound associated with any transition is at most c. Consider the event-clock automata A_{T_1} and A_{T_2}. Each automaton has $O(k^n)$ locations. The size of the alphabet is $O(n^2)$, and the clock constraints of A_{T_1} and A_{T_2} use constants bounded by c. Since the two automata are deterministic, to check if they accept the same timed language, we need to take the product, and construct the region automaton $Reg(A_{T_1} \otimes A_{T_2})$, and search for a path that is possible in one, but not in the other. The resulting automaton has $O(k^n \cdot 2^{O(n^2 \, log \, n^2 c)})$ regions. This implies the desired check can be performed in space polynomial in n and $log \, c$. ■

References

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

[2] R. Alur, C. Courcoubetis, and T. Henzinger. Computing accumulated delays in real-time systems. In *Proceedings of the Fifth Conference on Computer-Aided Verification*, LNCS 697, pages 181–193. Springer-Verlag, 1993.

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

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

[5] R. Alur and T. Henzinger. Back to the future: Towards a theory of timed regular languages. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 177–186, 1992.

[6] R. Alur, A. Itai, R. Kurshan, and M. Yannakakis. Timing verification by successive approximation. In *Proceedings of the Fourth Workshop on Computer-Aided Verification*, LNCS 663, pages 137–150. Springer-Verlag, 1992. To appear in Information and Computation.

[7] 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, 1991.

[8] D. 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.

[9] T. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proceedings of the 18th ACM Symposium on Principles of Programming Languages*, pages 353–366, 1991.

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

[11] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[12] R. Kurshan. *Computer-aided verification: the automata-theoretic approach.* Princeton University Press, 1994.

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

[14] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems.* Springer-verlag, 1991.

[15] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In *Proceedings of Workshop on Theories of Concurrency: CONCUR'91*, 1991.

[16] F. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. In *Real-Time: Theory in Practice, REX Workshop*, LNCS 600, pages 618–639. Springer-Verlag, 1991.

[17] A. Sistla, M. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[18] P. Wolper, M. Vardi, and A. Sistla. Reasoning about infinite computation paths. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 185–194, 1983.

[19] 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.

[20] T. Yoneda, A. Shibayam, B. Shlingloff, and E. Clarke. Efficient verification of parallel real-time systems. In *Proceedings of the Fifth Conference on Computer-Aided Verification*, LNCS 697, pages 321–332. Springer-Verlag, 1993.