Event-Clock Automata: A Determinizable Class of Timed Automata*

Rajeev Alur† Limor Fix‡ Thomas A. Henzinger§

Abstract. We introduce *event-recording automata.* An event-recording automaton is a timed automaton that contains, for every event a, a clock that records the time of the last occurrence of a. The class of event-recording automata 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 automata*, which contain clocks that predict the time of the next occurrence of an event. The class of *event-clock automata*, 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 [15, 21]. 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

*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.

†Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, and Bell Laboratories, Lucent Technologies, Murray Hill, NJ. Email: alur@research.bell-labs.com.

‡Design Technology, Intel Corporation, Haifa, Israel. Email: fix@iil.intel.com.

§Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA. Email: tah@eecs.berkeley.edu. Home page: www.eecs.berkeley.edu/~tah. Supported in part by the Office of Naval Research Young Investigator award N00014-95-1-0520, by the National Science Foundation CAREER award CCR-9501708, by the National Science Foundation grant CCR-9504469, by the Air Force Office of Scientific Research contract F49620-93-1-0056, by the Army Research Office MURI grant DAAH-04-96-1-0341, by the Advanced Research Projects Agency grant NAG2-892, and by the Semiconductor Research Corporation contract 95-DC-324.036.

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 [14, 20].

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*. With each transition, a timed automaton may check the clock values, and reassign new values to some 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 solution of certain verification problems for real-time systems with finite control [1, 3, 4, 6, 13], and the solution of certain delay problems [2, 9]. Solutions based on this theory have been implemented in several automatic tools, including COSPAN [7], KRONOS [10], and UPPAAL [8]. 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 real-time computation with finite control [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 (PSPACE-complete) for event-clock automata.

The class of event-clock automata is sufficiently expressive to model real-time systems with finite control, and to specify common real-time requirements. For instance, the hard real-time requirements that "every request is followed by a response within 3 seconds" and that "every two consecutive requests are separated by at least 5 seconds" can be expressed 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* [12]. 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 [18, 19]). 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. Section 3 proves that for every nondeterministic event-clock automaton, we can construct an equivalent deterministic event-clock automaton. Section 4 studies closure properties and decision problems of event-clock automata. Section 5 relates the expressiveness of various classes of timed automata. Section 6 shows how event-clock automata can be used 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 $t_i \in \mathbb{R}^{\geq 0}$ 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 $0 \leq i < n$). Without loss of generality it may be assumed that $t_0 = 0$. Sometimes we denote the timed word $\overline{w}$ by the pair $(\overline{a}, \overline{t})$, where $\overline{a} \in \Sigma^*$ is an untimed word over Σ. 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 $\mathit{Untime}(\mathcal{L})$ is the untimed language over Σ that is obtained by discarding the time-stamps: $\mathit{Untime}(\mathcal{L}) \subseteq \Sigma^*$ consists of all untimed words $\overline{a}$ 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 whose transitions 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 real-time systems with finite control. The finite control of a timed automaton consists of 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 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 previous 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 $1 \leq z \leq 2$ 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.

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

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 $\mathbb{R}_{\perp}^{\geq 0} = \mathbb{R}^{\geq 0} \cup \{\perp\}$ for the set of nonnegative real numbers together with the special value $\perp$. Formally, we define for all $0 \leq j \leq n$,

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

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 [23], and in assertional proof methods [16, 19].

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 difference between the current input symbol and the next occurrence of the input symbol a; the special value $\perp$ indicates the absence of a future occurrence of a. Formally, we define for all $0 \leq j \leq n$,

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

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 $\{x_a \mid a \in \Sigma\} \cup \{y_a \mid a \in \Sigma\}$ of event-recording and event-predicting clocks. For each position j of a timed word $\overline{w}$, the *clock-valuation function* $\gamma_j^{\overline{w}}$, then, is a mapping from C_Σ to $\mathbb{R}_{\perp}^{\geq 0}$. The clock constraints compare clock values to rational constants or to the special value $\perp$. Let $\mathbb{Q}_{\perp}^{\geq 0}$ 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 $c \in \mathbb{Q}_{\perp}^{\geq 0}$. The clock constraints over C are interpreted with respect to clock-valuation functions γ from C to $\mathbb{R}_{\perp}^{\geq 0}$: the atom $\perp \leq \perp$ evaluates to true, and all other comparisons that involve $\perp$ (e.g., $\perp \geq 3$) evaluate to false. For the 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. We freely use abbreviations such as $<$ and $=$ when writing clock constraints.

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.[2] Formally, an *event-clock automaton* A consists of

[2] Clock constraints can be added, as invariant conditions, also to the locations of an event-clock automaton [13], without influencing our results.

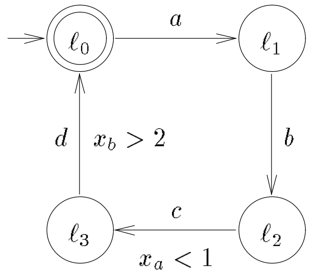

Figure 1: Event-recording automaton A_1

- 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, 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.

An (untimed) finite-state machine can be viewed as an event-clock automaton all of whose edges have the trivial clock constraint *true*, which evaluates to true for all clock-valuation functions.

Consider the behavior of the event-clock automaton A 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 if there is an edge $(\ell, \ell', a, \varphi)$ such that a equals the current input symbol a_i, and the current clock valuation $\gamma_i^{\overline{w}}$ 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 $0 \leq i \leq n$, $\gamma_i^{\overline{w}} \models \varphi_i$. The computation is *accepting* iff $\ell_{n+1} \in L^f$. The timed language $\mathcal{L}(A)$ defined by the event-clock automaton A 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* iff all clock constraints of A contain only event-recording clocks; A is an *event-predicting automaton* iff 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 automaton accepts timed

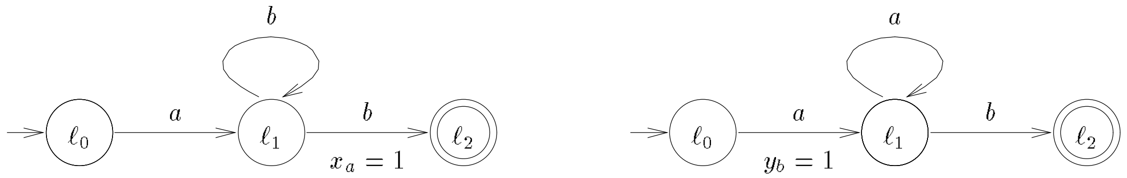

Figure 2: Event-recording automaton A_2 and event-predicting automaton A_3

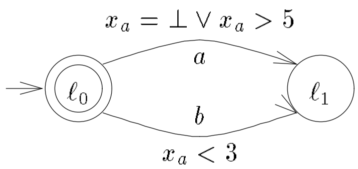

Figure 3: Event-recording automaton A_4

input words of the form $(\overline{a}, \overline{t})$ with $\overline{a} = (abcd)^k$, for some $k \geq 0$. All edges that are not labeled by clock constraints have, by default, the trivial clock constraint *true*. The clock constraint $x_a < 1$ that is associated with the edge from ℓ_2 to ℓ_3 ensures that each c occurs within 1 time unit of the preceding a. A similar mechanism for checking the value of x_b while reading d ensures that the time difference between each 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 $(\overline{a}, \overline{t})$ such that for all $0 \leq j < k$, we have $t_{4j+2} < t_{4j} + 1$ 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 will 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 $(aa^*b, \overline{t})$ 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 as the railroad-gate controller and timing-based mutual-exclusion algorithms that appear in the literature on real-time verification (see, for instance, [3, 6, 13]) can all be specified using event-clock automata.

3 Deterministic Event-clock Automata

A finite-state machine (with a single start location) is deterministic if all input symbols that label edges with the same source location are pairwise distinct. For event-clock automata we consider the notion of determinism that was proposed for timed automata [3]. The event-clock automaton

$A = (\Sigma, L, L^0, L^f, E)$ is *deterministic* iff

1. A has at most one start location (i.e., $|L^0| \leq 1$), 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, $\gamma \not\models \varphi_1 \wedge \varphi_2$.

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 timed 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 machine 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 subclass of timed automata for which the deterministic and nondeterministic automata are equally expressive.

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 nonempty subsets of L.
- The only start location is L^0 (if L^0 is empty, then $Det(A)$ has no start location).
- 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 edges of A whose source locations are in L' and whose input symbol is a. Then, for every nonempty subset E'' of E', there is an edge from L' to L'' 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 $(E' \setminus E'')$.

For example, Figure 4 shows a nondeterministic event-recording automaton A_5 and the determinized automaton $Det(A_5)$.

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

1. $\mathcal{L}(A) = \mathcal{L}(Det(A))$.

2. Given a location L' of $Det(A)$, an input symbol a, and a clock-valuation function γ, there is precisely one edge (L', L'', a, φ) such that $\gamma \models \varphi$. Hence, $Det(A)$ is deterministic.

3. $Det(A)$ is an event-recording (event-predicting) automaton iff A is an event-recording (event-predicting) automaton.

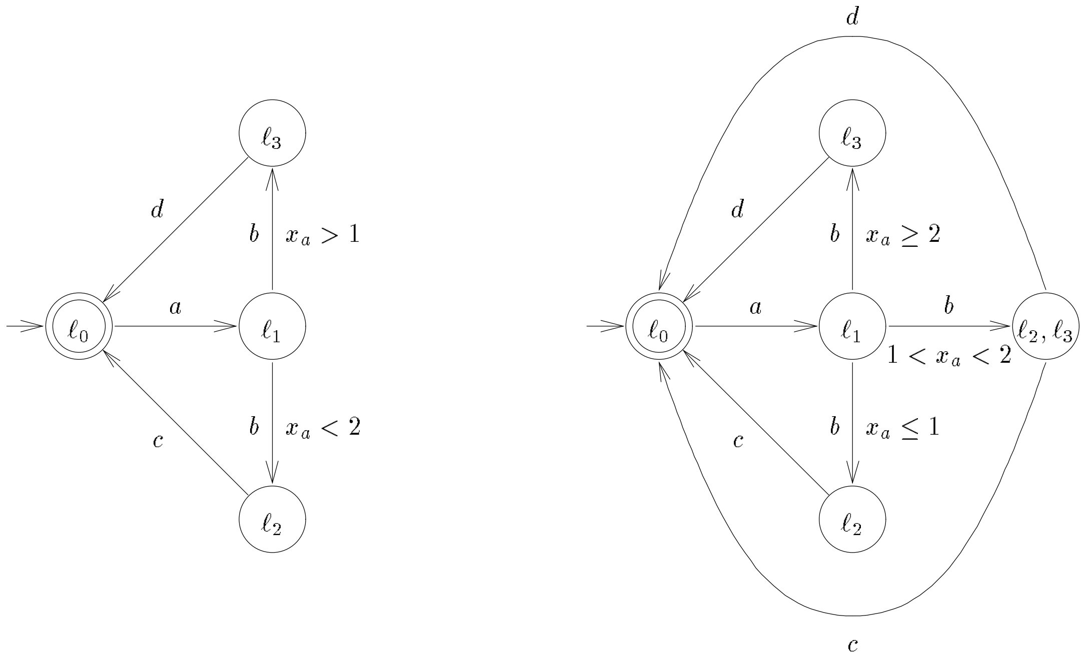

Figure 4: Event-recording automata A_5 and $Det(A_5)$

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 the timed language $\mathcal{L}(A)$.*

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 in the input word, 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 might want to use for each linear temporal logic formula ψ over the input alphabet 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. A *formula-clock automaton*, then, is a timed automaton all of whose clocks are either formula-recording or formula-predicting (event-clock automata are the subclass of formula-clock automata for which all formulas are atomic). Similarly to event-clock automata, every formula-clock automaton can be determinized.

4 Properties of Event-clock Automata

Event-clock automata as labeled transition systems

Every timed automaton can be viewed as an infinite-state labeled transition system. Given an event-clock automaton $A = (\Sigma, L, L^0, L^f, E)$, we define the labeled transition system S_A to capture the behavior of A over timed words. The labeled transition system S_A has the following components:

- A *state* of S_A is a pair (ℓ, γ) that consists of a location $\ell \in L$ and a clock-valuation function γ from C_Σ to $\mathbb{R}_\perp^{\geq 0}$, which determines the values of all clocks. The state space of S_A is denoted by Q_A.
- The set $Q_A^0 \subseteq Q_A$ of initial states consists of all states (ℓ, γ) such that $\ell \in L^0$, and $\gamma(x_a) = \perp$ for all input symbols $a \in \Sigma$.
- The set $Q_A^f \subseteq Q_A$ of final states consists of all states (ℓ, γ) such that $\ell \in L^f$, and $\gamma(y_a) = \perp$ for all input symbols $a \in \Sigma$.
- For two states $q, q' \in Q_A$, an input symbol $a \in \Sigma$, and a real-valued time delay $\delta \in \mathbb{R}^{\geq 0}$, let $q \xrightarrow{a} q'$ if the automaton A may proceed from the state q to the state q' by reading the input symbol a, and let $q \xrightarrow{\delta} q'$ if A may proceed from q to q' by letting time δ pass. Formally,
  - $(\ell, \gamma) \xrightarrow{a} (\ell', \gamma')$ iff there is a clock-valuation function 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 $\gamma'(x_b) = \perp$, otherwise $\gamma'(x_b) = \gamma(x_b) + \delta$, and
    - if $\gamma'(y_b) = \perp$ then $\gamma(y_b) = \perp$, otherwise $\gamma(y_b) = \gamma'(y_b) + \delta$.

We inductively extend the labeled transition relation to timed words:

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

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 $q \xrightarrow{\overline{w}} q'$ for some initial state q and some final state q' of the labeled transition system S_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 [3]. Here we apply the region construction to event-clock automata.

Consider an event-clock automaton A and the corresponding labeled transition system S_A. An equivalence relation $\cong$ on the state space Q_A is a *time-abstract bisimulation* of A iff for all states $q_1, q_2 \in Q_A$, if $q_1 \cong q_2$ then

1. if $q_1 \xrightarrow{a} q_1'$ for some input symbol $a \in \Sigma$, then there exists a state $q_2' \in Q_A$ with $q_2 \xrightarrow{a} q_2'$ and $q_1' \cong q_2'$, and
2. if $q_1 \xrightarrow{\delta} q_1'$ for some time delay $\delta \in \mathbb{R}^{\geq 0}$, then there exists a state $q_2' \in Q_A$ and a time delay $\delta' \in \mathbb{R}^{\geq 0}$ (possibly different from δ) with $q_2 \xrightarrow{\delta'} q_2'$ and $q_1' \cong q_2'$.

The relation $\cong$ has *finite index* iff the number of equivalence classes of $\cong$ is finite. Time-abstract bisimulations with finite index can be used to solve reachability problems for A. One such relation is the region equivalence $\cong_A$.

Let us 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 clock constraints). 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 $\mathbb{R}_\perp^{\geq 0}$ are *region-equivalent*, written $\gamma \cong_A \gamma'$, iff the following three conditions are satisfied:

1. γ and γ' agree on which clock values are undefined: for all $z \in C_\Sigma$, we have $\gamma(z) = \perp$ 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 $\lfloor \gamma(z) \rfloor = \lfloor \gamma'(z) \rfloor$ and $\lceil \gamma(z) \rceil = \lceil \gamma'(z) \rceil$.
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 $\lceil \gamma(y_a) \rceil - \gamma(y_a)$. Then: for all $z, z' \in C_\Sigma$, if $\gamma(z) \leq c$ and $\gamma(z') \leq c$, then $\langle \gamma(z) \rangle \leq \langle \gamma(z') \rangle$ iff $\langle \gamma'(z) \rangle \leq \langle \gamma'(z') \rangle$.

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

Lemma 2 [3] *For every event-clock automaton A with integer constants, the region-equivalence relation $\cong_A$ is a time-abstract bisimulation of A.*

An equivalence class of $\cong_A$ is called a *region* of A. The number of regions of A is finite.

Lemma 3 [3] *For every event-clock automaton A with integer constants, the number of regions of A is $n \cdot 2^{O(m \log cm)}$, where n is the number of locations of A, m is the size of the input alphabet, and c is the largest constant that appears in a clock constraint of A.*

Given the time-abstract bisimulation $\cong_A$ with finite index, we define the *region automaton* $Reg_{\cong}(A)$ as a finite-state machine over the input alphabet Σ, with the following components:

- The locations of $Reg_{\cong}(A)$ are the regions of A.

- A region is a start location iff it contains an initial state of S_A, and an accepting location iff it contains a final state of S_A.
- There is an edge from the region ρ to the region ρ' labeled with the input symbol $a \in \Sigma$ iff there are two states $q \in \rho$ and $q' \in \rho'$ of S_A, and a time delay $\delta \in \mathbb{R}^{\geq 0}$, such that $q \xrightarrow{(a,\delta)} q'$.

From Lemma 1 and the definition of time-abstract bisimulations, it follows that the region automaton $Reg_{\cong}(A)$ defines the language $Untime(\mathcal{L}(A))$.

Theorem 2 (Untiming) [3] *For every event-clock automaton A, the untimed language $Untime(\mathcal{L}(A))$ is regular.*

Closure properties

While the class of timed automata is not closed under complement, and the language-inclusion 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 $A_1 \times A_2$ is a pair consisting 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 renaming or hiding of input symbols. Consider the timed language $\mathcal{L}$ over a unary alphabet that contains all 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, neither by an event-clock automaton. This fact can be used to prove nonclosure properties. For instance, consider the timed language $\mathcal{L}'$ that contains all timed words of the form $(a^*ba^*ba^*, \overline{t})$ such that the time difference between 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 timed words $\overline{w} = (\overline{a}, \overline{t})$ over the unary alphabet $\{a\}$ 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 ERA and EPA are closed under complement, it follows that neither class is closed under renaming.

Similarly, consider the timed language $\mathcal{L}''$ that contains all timed words in $\mathcal{L}'$ such that both b-symbols are followed by a-symbols after exactly time 0.5. The timed language $\mathcal{L}''$ is in ERA. If we hide the input symbol b, the resulting timed language is again the complement of $\mathcal{L}$, which implies that ERA is not closed under hiding. An analogous argument applies to EPA.

Decision procedures

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 $Reg_{\cong}(A)$ and check if the untimed language of the finite-state machine $Reg_{\cong}(A)$ 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 of A with the smallest number of regions using minimization algorithms [22], or to incorporate the clock constraints of A one by one, generating successive approximations to the region automaton $Reg_{\cong}(A)$ [6]. Reachability problems for timed automata can also be solved by symbolic fixpoint computation [11, 13].

To check if the timed language of the event-clock automaton A_1 is included in the timed 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 timed 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.*[3]

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 input alphabet. The first step involves multiplying all constants in the clock constraints of A_1 and A_2 by the least common multiple of the denominators so that the clock constraints contain only integer constants, thus obtaining A_1' and A_2'. Let c be the largest integer constant that appears in the clock constraints after this normalization step. The length of c (i.e., the number of bits required to represent c) is at most quadratic in the length of the encoding of the original clock 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 integer constants that appear in the clock constraints of $\neg Det(A_2')$ are bounded by c. Let A be the product of A_1' and $\neg Det(A_2')$. The event-clock automaton A has $n \cdot 2^n$ locations, and the integer constants that appear in the clock constraints of A are also bounded by c. By Lemma 3, the region automaton $Reg_{\cong}(A)$ has $n \cdot 2^n \cdot 2^{O(m \log cm)}$ regions; that is, the number of regions is singly exponential in the length of the description of the input automata A_1 and A_2. Checking emptiness corresponds to searching for accepting paths in this exponential-sized finite-state machine. Since the rules that define the edges of $Reg_{\cong}(A)$ can be verified in polynomial time, it follows that emptiness can be checked in PSPACE.

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

The algorithm for language inclusion can be used to verify whether a system described as a timed automaton satisfies a specification given as an event-clock automaton.

5 Relating Classes of Timed Automata

Timed automata

We briefly review the definition of a timed automaton [3]. A *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 $L^f \subseteq L$

[3] In fact, A_1 may be any timed automaton.

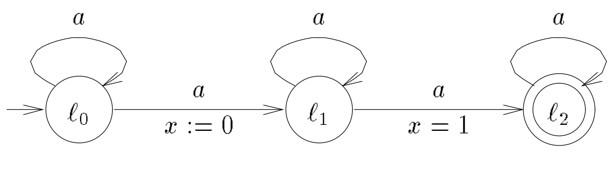

Figure 5: Timed automaton A_6

of accepting locations, a finite set C of clocks, and a finite set E of edges. Each edge $e \in E$ is a quintuple $(\ell, \ell', a, \varphi, \rho)$ with a source location $\ell \in L$, a target location $\ell' \in L$, an input symbol $a \in \Sigma$, a clock constraint φ over C, and a *reset condition* $\rho \subseteq C$ that specifies the clocks that are reset to 0 when the edge e is traversed. A clock-valuation function γ for the timed automaton A is a function from the clocks C to the extended reals $\mathbb{R}^{\geq 0}_{\perp}$. For a time delay $\delta \in \mathbb{R}^{\geq 0}$, we write $\gamma + \delta$ for the clock-valuation function that assigns to each clock $x \in C$ the value $\gamma(x) + \delta$. For a set $\rho \subseteq C$ of clocks, we write $\gamma' = \gamma[\rho := 0]$ for the clock-valuation function that agrees with γ on all clocks except those in ρ, which evaluate to 0 (i.e., $\gamma'(x) = \gamma(x)$ if $x \notin \rho$, and $\gamma'(x) = 0$ if $x \in \rho$). A *computation* of the timed automaton A over the timed input word $\overline{w} = (a_0, t_0) \ldots (a_n, t_n)$ 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-valuation functions γ_i, and edges $e_i = (\ell_i, \ell_{i+1}, a_i, \varphi_i, \rho_i) \in E$ such that

1. $\ell_0 \in L^0$ and for all clocks $x \in C$, we have $\gamma_0(x) = \perp$, and
2. for all $0 \leq i \leq n$, we have $\gamma_i + (t_i - t_{i-1}) \models \varphi_i$ and $\gamma_{i+1} = (\gamma_i + t_i - t_{i-1})[\rho_i := 0]$.

The computation is *accepting* iff $\ell_{n+1} \in L^f$. The timed language $\mathcal{L}(A)$ defined by the timed automaton A consists of all timed words $\overline{w}$ such that A has an accepting computation over $\overline{w}$. Figure 5 shows the timed automaton A_6, which uses a single clock x to accept timed words over the unary alphabet $\{a\}$. The timed language $\mathcal{L}(A_6)$ consists of all timed words of the form $(a^k, \overline{t})$, for $k \geq 2$, such that $t_j - t_i = 1$ for some $0 \leq i < j < k$.

We write NTA for the class of timed languages that are definable by timed automata. The class NTA is closed under union and intersection, but not under complement [3]. In particular, the complement of the language $\mathcal{L}(A_6)$, which contains all timed words in which no two symbols occur with time difference 1, cannot be accepted with finitely many clocks. Checking emptiness for timed automata is PSPACE-complete, while language inclusion for timed automata cannot be decided [3].

The definition of determinism for timed automata is the same as for event-clock automata; that is, a timed automaton is *deterministic* iff it has at most one start location, and two edges with the same source location and the same input symbol have mutually exclusive clock constraints. 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 [3].

From event-clock automata to timed automata

Every event-clock automaton can be translated into a timed automaton that defines the same timed language. Translating event-recording clocks is easy: an event-recording clock x_a is reset on an edge e iff the input symbol of e is a. This trivial translation preserves determinism. The translation of event-predicting clocks introduces nondeterminism.

Consider an event-predicting automaton $A = (\Sigma, L, L^0, L^f, E)$. An *atomic clock constraint* is a formula of the form $y_a = \perp$ or $y_a \sim c$, where $\sim$ stands for $\leq$ or $<$ or $\geq$ or $>$. We assume that the clock constraint of each edge of A is a conjunction of atomic clock constraints; this can always be achieved by writing clock constraints in disjunctive normal form and creating separate edges for all disjuncts. Let Φ_A be the set of atomic clock constraints that appear in the edges of A. We construct a nondeterministic timed automaton B over the input alphabet Σ as follows:

- The locations of B are the pairs (ℓ, Ψ) with $\ell \in L$ and $\Psi \subseteq \Phi_A$.
- The location (ℓ, Ψ) is a start location of B iff $\ell \in L^0$ and Ψ does not contain a constraint of the form $y_a \sim c$.
- The location (ℓ, Ψ) is an accepting location of B iff $\ell \in L^f$ and Ψ equals $\{y_a = \perp \mid a \in \Sigma\}$.
- For every constraint $\psi \in \Phi_A$, the automaton B has a clock z_ψ.
- The automaton B has an edge from the source location (ℓ, Ψ) to the target location (ℓ', Ψ') with the input symbol a, the clock constraint φ, and the reset condition ρ iff the following seven conditions are met. 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 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 of the form (ℓ, ℓ', a, χ).
  2. The constraint $y_a = \perp$ does not appear in Ψ.
  3. The constraint φ is the conjunction of all atomic clock constraints of the form $(z_{(y_a \sim c)} \sim c)$ with $(y_a \sim c) \in \Psi$.
  4. For each input symbol b different from a, if a constraint involving y_b appears in Ψ, then it appears in Ψ' also.
  5. Each conjunct of χ appears in Ψ' also.
  6. For each input symbol b and for $\sim$ equal to $>$ or $\geq$, the clock $z_{(y_b \sim c)}$ appears in the reset condition ρ iff the constraint $y_b \sim c$ is a conjunct of χ.
  7. For each input symbol b and for $\sim$ equal to $<$ or $\leq$, the clock $z_{(y_b \sim c)}$ appears in the reset condition ρ iff the constraint $y_b \sim c$ is a conjunct of χ, and either $b = a$ or the constraint $y_b \sim c$ does not appear in Ψ.

Then the timed automaton B defines the timed language $\mathcal{L}(A)$.

The following theorem relates the various classes of timed automata. The relationships are also shown in Figure 5.

Theorem 5 (Relationships between classes of timed automata)

(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		

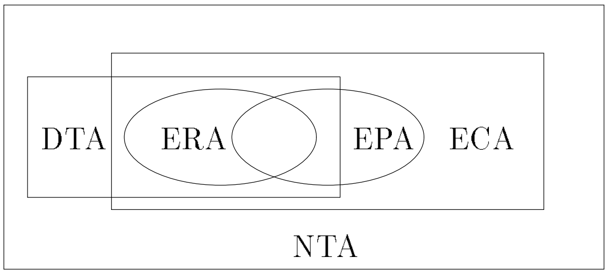

Figure 6: Relationships between classes of timed automata

Proof. For (1), the timed language of the event-recording automaton A_2 of Figure 2 is not definable by an event-predicting automaton. The proof is by contradiction. Suppose that 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 the 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 the two clocks y_a and y_b. Although the clock-valuation function $\gamma_1^{\overline{w}_1}(y_b)$ differs from the clock-valuation function $\gamma_1^{\overline{w}_2}(y_b)$, clock constraints with integer constants cannot detect this difference: for every clock constraint φ of B and every position $0 \leq j \leq 2$, we have $\gamma_j^{\overline{w}_1} \models \varphi$ iff $\gamma_j^{\overline{w}_2} \models \varphi$. 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 timed language of the event-predicting automaton A_3 of Figure 2 cannot be defined by an event-recording automaton. The proof is similar to case (1).

For (3), consider the union of the two automata A_2 and A_3. The resulting automaton is an event-clock automaton. A proof similar to 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 $\mathcal{L} = \{(a^k b, t_0 \ldots t_k) \mid \exists 0 \leq j < k.\, t_k - t_j = 1\}$ 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 that a deterministic timed automaton B defines the timed language $\mathcal{L}$. Assume that B uses only integer constants, and m clocks. Consider the timed word $\overline{w} = (a^{m+2}, t_0 \ldots t_{m+1})$ with $0 = t_0 < t_1 < \cdots < t_m < t_{m+1} = 1$. Since B is deterministic, there is at most one computation of B that reads the word $\overline{w}$. Since B has at most m clocks, there is at least one position $1 \leq j \leq m$ such that no clock of B has the value $1 - t_j$ when that computation reads the input $(a, 1)$. Hence, the automaton B "forgets" the time-stamp t_j. Consider two extensions of the timed word $\overline{w}$: let $\overline{w}_1$ be $\overline{w}$ followed by $(b, t_j + 1)$, and let $\overline{w}_2$ be $\overline{w}$ followed by $(b, t' + 1)$, where $t' \neq t_j$ is chosen such that $t_{j-1} < t' < t_{j+1}$. The clocks of B satisfy the same clock constraints when reading the additional, $(m+2)$-nd input symbol in both cases. Thus, the automaton B accepts $\overline{w}_1$ iff it accepts $\overline{w}_2$. But, but $\overline{w}_1$ is in the timed language $\mathcal{L}$, 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 timed automaton with one clock x simply resets x when reading the first a, and checks the clock constraint $x = 1$ when reading the third a) but not in ECA. The proof is similar to 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 it rejects both timed words. ∎

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 timed automata that can read the timed input word a bounded number of times (by moving forward and backward over the input). 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 timed input words, 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 more suited for specification than deterministic two-way timed automata.

6 Timed Transition Systems as Event-clock Automata

Timed transition systems

A *transition system* T consists of a set Q of states, a set $Q^0 \subseteq Q$ of initial states, and a finite set $\mathcal{T}$ of transitions. Each transition $\tau \in \mathcal{T}$ is a function from Q to 2^Q: for each state $q \in Q$, the set $\tau(q)$ gives the possible τ-successors of q. The transition system T is *finite* iff the set Q of states is finite. A *run* $\overline{q}$ of the transition system T is a finite sequence $q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_n$ of states such that $q_0 \in Q^0$ and for all $0 \leq i < n$, there exists a transition $\tau_i \in \mathcal{T}$ with $q_{i+1} \in \tau_i(q_i)$. The transition τ is *enabled* at the i-th step of the run $\overline{q}$ iff $\tau(q_i)$ is nonempty, and τ is *taken* at the i-th step iff $q_i \in \tau(q_{i-1})$ (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 [17].

The model of transition systems is extended to timed transition systems so that it is possible to express real-time constraints on the transitions [12]. A *timed transition system* T consists of a transition system $(Q, Q^0, \mathcal{T})$ and two functions l and u from $\mathcal{T}$ to $\mathbb{Q}^{\geq 0}$ 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 q_i to the state q_{i+1} at time t_{i+1}. A *timed run* r of the timed transition system T is a finite sequence

$$\xrightarrow{t_0} q_0 \xrightarrow{t_1} q_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} q_n$$

of states $q_i \in Q$ and nondecreasing time-stamps $t_i \in \mathbb{R}^{\geq 0}$ such that $\overline{q}$ is a run of the underlying transition system and the following two conditions are met:

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* iff they have the same timed runs.

From timed transition systems to event-recording automata

We 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 r with states q_i and time-stamps t_i, we associate the timed word

$$\overline{w}_r \;=\; (\langle \perp, q_0\rangle, t_0)\ (\langle q_0, q_1\rangle, t_1)\ (\langle q_1, q_2\rangle, t_2)\ \ldots\ (\langle q_{n-1}, q_n\rangle, t_n),$$

where $\perp$ is a special symbol not in Q (as usual, $Q_\perp = Q \cup \{\perp\}$). Notice that the timed run r and the corresponding timed word $\overline{w}_r$ contain the same information: each event (i.e., state change) of r is modeled by a pair of states—a source state and a target state. Every finite timed transition system $T = (Q, Q^0, \mathcal{T}, l, u)$, then, defines a timed language $\mathcal{L}(T)$ over the alphabet $Q_\perp \times Q$, namely, the set of timed words $\overline{w}_r$ that correspond to timed runs r of T. Furthermore, 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 $q \in Q$ and, for each transition $\tau \in \mathcal{T}$, a pair of states $\langle \alpha(\tau), \beta(\tau)\rangle \in Q_\perp \times Q$ such that if τ is enabled in q, 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 each initial state $q_0 \in Q^0$, there is an edge from the source location ℓ_0 to the target location $(q_0, \langle \alpha, \beta\rangle)$ with the input symbol $\langle \perp, q_0\rangle$ and the trivial clock constraint *true*, where $\langle \alpha(\tau), \beta(\tau)\rangle = \langle \perp, q_0\rangle$ for all transitions $\tau \in \mathcal{T}$. In addition, there is an edge from the source location $(q, \langle \alpha, \beta\rangle)$ to the target location $(q', \langle \alpha', \beta'\rangle)$ with the input symbol $\langle q, q'\rangle$ and the conjunction φ of atomic clock constraints iff there is a transition $\tau \in \mathcal{T}$ such that $(q, q') \in \tau$, and for all transitions $\tau \in \mathcal{T}$,

1. if τ is enabled in q and $q' \notin \tau(q)$, then $\langle \alpha'(\tau), \beta'(\tau)\rangle = \langle \alpha(\tau), \beta(\tau)\rangle$, else $\langle \alpha'(\tau), \beta'(\tau)\rangle = \langle q, q'\rangle$,
2. if τ is enabled in q, then φ contains the conjunct $x_{\langle \alpha(\tau), \beta(\tau)\rangle} \leq u(\tau)$, and
3. if $q' \in \tau(q)$, 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. Consider two finite timed transition systems T_1 and T_2. Suppose that each transition system has at most n states and m transitions, and the bounds associated with the transitions are at most c. Consider the event-clock automata A_{T_1} and A_{T_2}. Each automaton has $2^{O((m+1)\log n)}$ locations. The size of the input alphabet is $(n+1) \cdot n$, 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 their product, construct the region automaton $Reg_{\cong}(A_{T_1} \times A_{T_2})$, and search for a path that is accepting in one component, but not in the other. The resulting region automaton has $2^{O((m+1)\log n + n^2 \log cn)}$ many regions. This implies that the desired check can be performed in space polynomial in n and m and $\log c$. The lower bound follows from the fact that it is PSPACE-hard to check if two nondeterministic finite-state machines accept the same language. ■

References

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

[2] R. Alur, C. Courcoubetis, and T. Henzinger. Computing accumulated delays in real-time systems. In *Proceedings of the Fifth Annual 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. *Journal of the ACM*, 43:116–146, 1996.

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

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

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

[8] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL: a tool suite for automatic verification of real-time systems. In *Hybrid Systems III: Verification and Control*, LNCS 1066, pages 232–243. Springer-Verlag, 1996.

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

[10] 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.

[11] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the First Annual Conference on Computer-Aided Verification*, LNCS 407, pages 197–212. Springer-Verlag, 1989.

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

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

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

[15] R. Kurshan. *Computer-Aided Verification: The Automata-Theoretic Approach.* Princeton University Press, 1994.

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

[17] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1991.

[18] M. Merritt, F. Modugno, and M. Tuttle. Time-constrained automata. In *Proceedings of the Second Annual Conference on Concurrency Theory*, LNCS 527, pages 408–423. Springer-Verlag, 1991.

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

[20] 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.

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

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

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