Decision Problems for Timed Automata: A Survey*

Rajeev Alur and P. Madhusudan

University of Pennsylvania

Abstract. Finite automata and regular languages have been useful in a wide variety of problems in computing, communication and control, including formal modeling and verification. Traditional automata do not admit an explicit modeling of time, and consequently, *timed automata* [2] were introduced as a formal notation to model the behavior of real-time systems. Timed automata accept *timed languages* consisting of sequences of events tagged with their occurrence times. Over the years, the formalism has been extensively studied leading to many results establishing connections to circuits and logic, and much progress has been made in developing verification algorithms, heuristics, and tools. This paper provides a survey of the theoretical results concerning decision problems of reachability, language inclusion and language equivalence for timed automata and its variants, with some new proofs and comparisons. We conclude with a discussion of some open problems.

1 Timed Automata

A timed automaton is a finite automaton augmented with a finite set of (real-valued) *clocks*. The vertices of the automaton are called *locations*, and edges are called *switches*. While switches are instantaneous, time can elapse in a location. A clock can be reset to zero simultaneously with any switch. At any instant, the reading of a clock equals the time elapsed since the last time it was reset. With each switch we associate a clock constraint, and require that the switch may be taken only if the current values of the clocks satisfy this constraint. Timed automata accept (or, equivalently, generate) timed words, that is, strings of symbols tagged with occurrence times. Let $\mathbb{R}$ denote the set of nonnegative real numbers, and let $\mathbb{Q}$ denote the set of nonnegative rational numbers. A *timed word* over an alphabet Σ is a sequence $(a_0, t_0), (a_1, t_1) \cdots (a_k, t_k)$, where each $a_i \in \Sigma$, each $t_i \in \mathbb{R}$, and the occurrence times increase monotonically: $t_0 \leq t_1 \leq \cdots \leq t_k$. The set of all timed words over Σ is denoted $T\Sigma^*$. A *timed language* over Σ is a subset of $T\Sigma^*$.

The *untimed* word corresponding to a timed word $(a_0, t_0), (a_1, t_1) \cdots (a_k, t_k)$ is the word $a_0 a_1 \ldots a_k$ obtained by deleting the occurrence times. The untimed language *untime*(L) of a timed language L consists of all the untimed words corresponding to the timed words in L. For an alphabet Σ, we use Σ^ϵ to denote

* This research was partially supported by NSF award ITR/SY 0121431.

$\Sigma \cup \{\epsilon\}$ (where ϵ is not in Σ), and for a subset $\Sigma' \subseteq \Sigma$, and a timed word $w = (a_0, t_0), (a_1, t_1) \cdots (a_k, t_k)$ over Σ, the projection of w over Σ' is obtained from w by deleting all (a_i, t_i) such that $a_i \notin \Sigma'$. The projection operation extends to timed languages as well.

To define timed automata formally, we need to say what type of clock constraints are allowed as guards. For a set X of clocks, the set $\Phi(X)$ of *clock constraints* g is defined by the grammar

$$g := x \leq c \mid c \leq x \mid x < c \mid c < x \mid g \wedge g$$

where $x \in X$ and $c \in \mathbb{Q}$. A *clock valuation* ν for a set X of clocks assigns a real value to each clock; that is, it is a mapping from X to $\mathbb{R}$. For $\delta \in \mathbb{R}$, $\nu + \delta$ denotes the clock valuation which maps every clock x to the value $\nu(x) + \delta$. For $Y \subseteq X$, $\nu[Y := 0]$ denotes the clock valuation for X which assigns 0 to each $x \in Y$, and agrees with ν over the rest of the clocks.

A *timed automaton* A over an alphabet Σ is a tuple $\langle V, V^0, V^F, X, E\rangle$, where

- V is a finite set of locations,
- $V^0 \subseteq V$ is a set of initial locations,
- $V^F \subseteq V$ is a set of final locations,
- X is a finite set of clocks,
- $E \subseteq V \times \Sigma^\epsilon \times \Phi(X) \times 2^X \times V$ is a set of switches. A switch $\langle s, a, g, \lambda, s'\rangle$ represents an edge from location s to location s' on symbol a. The *guard* g is a clock constraint over X that specifies when the switch is enabled, and the *update* $\lambda \subseteq X$ gives the clocks to be reset to 0 with this switch.

The semantics of a timed automaton A is defined by associating an infinite-state automaton S_A over the alphabet $\Sigma \cup \mathbb{R}$. A state of S_A is a pair (s, ν) such that s is a location of A and ν is a clock valuation for X. A state (s, ν) is an initial state if s is an initial location (i.e. $s \in V^0$) and $\nu(x) = 0$ for all clocks x. A state (s, ν) is a final state if s is a final location (i.e. $s \in V^F$). There are two types of transitions in S_A:

Elapse of time: for a state (s, ν) and a time increment $\delta \in \mathbb{R}$, $(s, \nu) \xrightarrow{\delta} (s, \nu + \delta)$.

Location switch: for a state (s, ν) and a switch $\langle s, a, g, \lambda, s'\rangle$ such that ν satisfies the guard g, $(s, \nu) \xrightarrow{a} (s', \nu[\lambda := 0])$.

For a timed word $w = (a_0, t_0), (a_1, t_1) \cdots (a_k, t_k)$ over Σ^ϵ, a *run* of A over w is a sequence

$$q_0 \xrightarrow{t_0} q_0' \xrightarrow{a_0} q_1 \xrightarrow{t_1 - t_0} q_1' \xrightarrow{a_1} q_2 \xrightarrow{t_2 - t_1} q_2' \xrightarrow{a_2} q_3 \rightarrow \cdots \xrightarrow{a_k} q_{k+1}$$

such that q_0 is an initial state of S_A. The run is *accepting* if q_{k+1} is a final state of S_A. The timed automaton A accepts a timed word w over Σ if there exists a timed word w' over Σ^ϵ such that A has an accepting run over w' and the projection of w' to Σ is w. The set of timed words accepted by A is denoted $L(A)$.

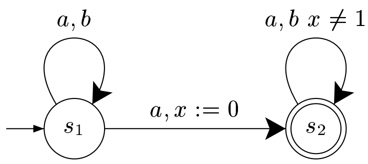

Fig. 1. A non-complementable timed automaton

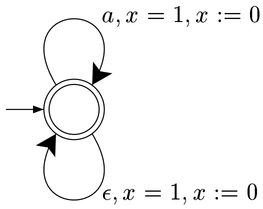

Fig. 2. ϵ-transitions increase expressiveness

A timed language $L \subseteq T\Sigma^*$ is said to be *timed regular* if there exists a timed automaton A such that $L(A) = L$. The closure properties of timed regular languages are summarized below:

Theorem 1. *The set of timed regular languages is closed under union, intersection, and projection, but not under complementation [2].*

The closure under union and intersection is established by extending the classical product construction to timed automata. Closure under projection is immediate since switches can be labeled with ϵ.

For the non-closure under complementation, we give a new proof here. Let $\Sigma = \{a, b\}$. Let L be the timed language consisting of timed words w containing an a event at some time t such that no event occurs at time $t + 1$. The (non-deterministic) timed automaton shown in Figure 1 (with initial location s_1 and final location s_2) accepts L.

We claim that $\overline{L}$, the complement of L, is not timed regular. Consider the timed language L' consisting of timed words w such that the untimed word of w is in a^*b^*, all the a events happen before time 1, and no two a events happen at the same time. Verify that L' is timed regular. Observe that a word of the form $a^n b^m$ belongs to $untime(\overline{L} \cap L')$ iff $m \geq n$. Since timed regular languages are closed under intersection, the untimed language of a timed regular language is regular (see Section 2), and the language $\{a^n b^m \mid m \geq n\}$ is not regular, it follows that $\overline{L}$ is not timed regular.

Unlike classical automata, ϵ-labeled switches add to the expressive power of timed automata [10]. For example, the automaton of Figure 2 accepts timed words w over $\{a\}$ such that every occurrence time is an integer and no two a-

events occur at the same time. This language cannot be accepted by a timed automaton if ϵ-labeled switches are disallowed: if the largest constant in a timed automaton A is c and A does not have ϵ-labeled switches, then A cannot distinguish between the words $(a, c+1)$ and$(a, c+1.1)$.

The more recent definitions of timed automata also admit labeling of each location with a clock constraint called its *invariant*, and require that time can elapse in a location only as long as its invariant stays true [23]. While this is a useful modeling concept to enforce upper bounds (without introducing "error" locations), it does not add to the expressive power.

Timed languages can also be defined using *timed state sequences*: a timed state sequence is a mapping from a prefix of the reals to a finite alphabet that can be represented by a sequence $(a_o, I_0)(a_1, I_1)\ldots(a_k, I_k)$, where $I_0, I_1, \ldots I_k$ is a sequence of adjoining intervals (e.g. $[0, 1.1)[1.1, 1.2](1.2, 1.7)$). Timed state sequences can be generated by timed automata in which locations are labeled with observations [23, 3]. This dual view does not change the core results, but some expressiveness results do differ in the two views [34].

2 Reachability and Language Emptiness

2.1 Region Automata

Given a timed automaton A, to check whether the language $L(A)$ is empty, we must determine if some final state is reachable from an initial state in the infinite-state system S_A. The solution to this reachability problem involves construction of a finite quotient. The construction uses an equivalence relation on the state-space that equates two states with the same location if they agree on the integral parts of all clock values and on the ordering of the fractional parts of all clock values. The integral parts of the clock values are needed to determine whether or not a particular clock constraint is met, whereas the ordering of the fractional parts is needed to decide which clock will change its integral part first. This is formalized as follows. First, assume that all the constants in the given timed automaton A are integers (if A uses rational constants, we can simply multiply each constant with the least-common-multiple of all the denominators to get an automaton with the same timed language modulo scaling). For any $\delta \in \mathbb{R}$, $\langle\delta\rangle$ denotes the fractional part of δ, and $\lfloor\delta\rfloor$ denotes the integral part of δ; $\delta = \lfloor\delta\rfloor + \langle\delta\rangle$. For each clock $x \in X$, let c_x be the largest integer c such that x is compared with c in some clock constraint appearing in a guard. The equivalence relation $\cong$, called the *region equivalence*, is defined over the set of all clock valuations for X. For two clock valuations ν and μ, $\nu \cong \mu$ iff all the following conditions hold:

1. For all clocks $x \in X$, either $\lfloor\nu(x)\rfloor$ and $\lfloor\mu(x)\rfloor$ are the same, or both $\nu(x)$ and $\mu(x)$ exceed c_x.
2. For all clocks x, y with $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, $\langle\nu(x)\rangle \leq \langle\nu(y)\rangle$ iff $\langle\mu(x)\rangle \leq \langle\mu(y)\rangle$.
3. For all clocks $x \in X$ with $\nu(x) \leq c_x$, $\langle\nu(x)\rangle = 0$ iff $\langle\mu(x)\rangle = 0$.

A *clock region* for A is an equivalence class of clock valuations induced by $\cong$. Note that there are only a finite number of regions, at most $k! \cdot 4^k \cdot \Pi_{x \in X}(c_x + 1)$, where k is the number of clocks. Thus, the number of clock regions is exponential in the encoding of the clock constraints.

The key property of region equivalence is its stability: for any location s, and clock valuations ν and ν' such that $\nu \cong \nu'$, (a) for any $\delta \in \mathbb{R}$, if $(s, \nu) \xrightarrow{\delta} (s, \nu + \delta)$ then there exists $\delta' \in \mathbb{R}$ such that $(s, \nu') \xrightarrow{\delta'} (s, \nu' + \delta')$ and $(\nu + \delta) \cong (\nu' + \delta')$, and (b) for every label $a \in \Sigma^\epsilon$ and state (t, μ), if $(s, \nu) \xrightarrow{a} (t, \mu)$ then there exists μ' such that $(s, \nu') \xrightarrow{a} (t, \mu')$ and $\mu \cong \mu'$. Thus, if two states are equivalent, then an a-labeled discrete switch from one can be matched by a corresponding discrete switch from the other leading to an equivalent target state, and if the automaton can wait for δ units in one state, then it can wait for δ' units, possibly different from δ, resulting in equivalent states. For this reason, the region equivalence is a *time-abstract* bisimulation.

For a timed automaton A, the quotient of S_A with respect to the region equivalence is called the *region automaton* of A, and is denoted $R(A)$: vertices of $R(A)$ are of the form (s, r), where s is a location and r is a clock region; there is an edge $(s, r) \xrightarrow{a} (s', r')$ in $R(A)$ for $a \in \Sigma^\epsilon$ iff for some clock valuations $\nu \in r$ and $\nu' \in r'$, $(s, \nu) \xrightarrow{a} (s', \nu')$ in S_A, or, $a = \epsilon$ and $(s, \nu) \xrightarrow{\delta} (s', \nu')$ for some $\delta \in \mathbb{R}$. The initial and final states of S_A are used to define the initial and final vertices of $R(A)$. Now, the language of $R(A)$ is the untimed language of $L(A)$.

Theorem 2. *For a timed regular language L, untime(L) is a regular language [2].*

Consequently, $R(A)$ can be used to solve language emptiness for A, and also to answer reachability queries for A. Thus, emptiness and reachability can be solved in time linear in the number of vertices and edges of the region automaton, which is linear in the number of locations and edges of A, exponential in the number of clocks, and exponential in the encoding of the constants. Technically, these problems are PSPACE-complete.

Theorem 3. *The language emptiness question for timed automata is* PSPACE-*complete, and can be solved in time $O(m \cdot k! \cdot 4^k \cdot (c \cdot c' + 1)^k)$, where m is the number of switches in A, k is the number of clocks in A, c is largest numerator in the constants in the clock constraints in A, and c' is the least-common-multiple of the denominators of all the constants in the clock constraints of A [2].*

In [15] it was also shown that for timed automata with three clocks, reachability is already PSPACE-complete. A recent result [28] shows that for timed automata with one clock, reachability is NLOGSPACE-complete and for timed automata with two clocks, it is NP-hard. The reachability problem remains PSPACE-hard even if we bound the magnitudes of constants [15].

2.2 Cycle detection

A timed ω-word is an infinite sequence of the form $\alpha = (a_0, t_0)(a_1, t_1) \ldots (a_i, t_i), \ldots$, with $a_i \in \Sigma$, $t_i \in \mathbb{R}$, and $t_0 \leq t_1 \leq \cdots t_i \leq \cdots$, and timed ω-language is a set of timed ω-words. Reasoning in terms of infinite timed words, as in the untimed setting, is useful for checking liveness properties. The notion of a run of a timed automaton A naturally extends to timed ω-words. A timed ω-word α is accepted by A using the Büchi condition, if there is a run of A on α that repeatedly hits (infinitely often) some final location in V^F. The set of ω-words accepted by A is denoted by $L_\omega(A)$. Checking whether $L_\omega(A)$ is nonempty, for a given A, can be done by checking whether there is a cycle in the region graph of A which is reachable from an initial state and contains some state in V^F.

For infinite words, it is natural to require that time diverges, that is, the sequence $t_0, t_1, \ldots t_i, \ldots$ grows without bound. Timed words that do not diverge depict an infinite number of events that occur in a finite amount of time. To restrict $L_\omega(A)$ only to divergent words, we can transform the timed automaton by adding a new clock x which is reset to 0 whenever it becomes 1 (using an ϵ-edge) and the timed automaton hits the new final set V'_F only if the run had passed through V_F in the last one unit of time.

Theorem 4. *Given a timed automaton A, the problem of checking emptiness of $L_\omega(A)$ is* PSPACE*-complete.*

Most of the results in this survey hold for timed ω-languages also.

2.3 Sampled Semantics

In the discrete-time or sampled semantics for timed automata, the discrete switches, or the events, are required to occur only at integral multiples of a given sampling rate f. This can be formalized as follows. Given a timed automaton A and a sampling rate $f \in \mathbb{Q}$, we define an automaton S_A^f: the states, initial states and final states of S_A^f are the same as the states, initial states, and final states of S_A, and the transitions of S_A^f are the transitions of S_A that are labeled with either $a \in \Sigma^\epsilon$ or with $m.f$ (where $m \in \mathbb{N}$). The sampled timed language $L^f(A)$ is defined using the automaton S_A^f. Note that time of occurrence of any symbol in the timed words in $L^f(A)$ is an integral multiple of the sampling frequency f. To check emptiness of $L^f(A)$, observe that in any reachable state of S_A^f, the values of all clocks are integral multiples of f, and this can lead to a reduced search space compared to the region automata. However, the complexity class of the reachability and cycle-detection problems stays unchanged (here L_ω^f denotes the set of ω-words where events occur at sampling rate f):

Theorem 5. *Given a timed automaton A and a sampling rate $f \in \mathbb{Q}$, the problem of checking the emptiness of $L^f(A)$ (or $L_\omega^f(A)$) is* PSPACE*-complete.*

If the sampling rate f is unknown, the resulting problems are the discrete-time reachability and discrete-time cycle-detection problems with unknown sampling rate: given a timed automaton A, does there exist a rational number $f \in \mathbb{Q}$ such that $L^f(A)$ (or $L_\omega^f(A)$) is nonempty. Discrete-time reachability for unknown sampling rate is decidable since it is equivalent to the question of whether $L(A)$

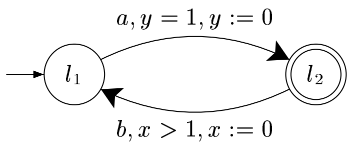

Fig. 3. Sampled semantics is different from the standard semantics

is empty: if $L(A)$ is nonempty, we can find a word in $L(A)$ where events occur at rational times, and by choosing an appropriate f, show that it is an f-sampled word. However, the discrete-time cycle-detection problem with unknown sampling rate is undecidable:

Theorem 6. *Given A, the problem of checking whether $\bigcup_{f \in \mathbb{Q}} L^f_\omega(A)$ is nonempty, is undecidable [14].*

The undecidability proof is by reduction from the halting problem for two-counter machines. Given a two-counter machine M, one can construct a timed automaton A_M and a location s_F such that for any integer n, the location s_F is reachable in the discrete-time semantics with the sampling rate $1/n$ iff the two-counter machine M has a halting run in which both the counters do not exceed the value n.

To see that $L_\omega(A)$ can be nonempty while for each f, $L^f_\omega(A) = \emptyset$, consider the automaton in Figure 3. While the a-events occur at integer times, the b-events have to occur closer and closer to the a-events, and fixing any sampling rate f makes the ω-language empty.

2.4 Choice of Clock Constraints and Updates

The clock constraints in the guards of a timed automaton compare clocks with constants. Such constraints allow us to express (constant) lower and upper bounds on delays. Consider the following generalization of clock constraints: for a set X of clocks, the set $\Phi^d(X)$ of *clock constraints* g is defined by the grammar

$$g := x \leq c \mid c \leq x \mid x - y \leq c \mid x < c \mid c < x \mid x - y < c \mid g \wedge g$$

where x, y are clocks in X and $c \in \mathbb{Q}$. Including such "diagonal" clock constraints that compare clock differences with constants does not change the complexity of reachability. Similarly, we can relax the allowed updates on switches. In the original definition, each switch is tagged with a set λ which specifies which clocks should be reset to zero. A more general *update map* λ maps clocks in X to $\mathbb{Q} \cup X$ specifying the assignments $x := \lambda(x)$. Thus, x can be assigned to an arbitrary rational constant, or to the value of another clock. Both these modifications can be handled by modifying the region construction. In fact, both these extensions do not add to the expressive power.

Theorem 7. *If the clock constraints for guards are chosen from the set $\Phi^d(X)$, and the switches are annotated with the update maps, the expressive power of timed automata stays unchanged, and the language emptiness problem stays* PSPACE*-complete.*

However, a variety of extensions have been shown to allow definition of languages that are not timed regular, and lead to undecidability of the emptiness problem. We summarize some notable ones:

1. Allowing guards of the form $x = 2y$ renders reachability problem for timed automata undecidable [2].
2. Allowing guards of the form $x + y \sim c$, where $\sim \in \{<, \leq\}$ leads to undecidability if there are four or more clocks, but is decidable for automata with two clocks [9].
3. Allowing updates of the form $x := x - 1$ renders reachability problem for timed automata undecidable [13].
4. Allowing updates of the form $x := x + 1$ keeps the reachability problem PSPACE-complete if the clock constraints are chosen from $\Phi(X)$, but renders it undecidable if the guards are chosen from $\Phi^d(X)$ [13].
5. Allowing guards that compare clocks with irrational constants renders reachability problem for timed automata undecidable [30].

The first result above implies that allowing constraints involving addition of clock variables leads to undecidability of the reachability problem. With an enabling condition of the form $y = 2x$, one can express a constraint of the kind "the time delay between the symbols a and b is the same as the time delay between b and c" (reset a clock x while reading a, reset a clock y while reading b, and require $y = 2x$ while reading c). This can be exploited to copy the counter values, and encode configurations of a two-counter machine, leading to undecidability. The second result is of similar nature. The third result says that one cannot allow decrements. Since clocks increase with elapse of time, with decrements they can act as counters, and thus be used to encode counter machines. The fourth result says that explicit increments can be allowed in the original region construction, but in the presence of guards of the "diagonal" form $x - y \leq c$, such increments allow encoding of counter values using the differences between clocks. Bouyer et al have also studied nondeterministic updates (for example, x is reset to a value chosen nondeterministically from intervals such as $[0, c]$ or $[y, \infty)$), and their impact on the decidability with and without the "diagonal" constraints [13]. Finally, [30] considers timed automata where guard constraints compare clocks with *irrational constants*, and shows that if $\tau \in (0, 1)$ is an irrational number, then timed automata where the constants are taken from $\{0, 1, \tau, 3 - \tau\}$ have an undecidable emptiness problem.

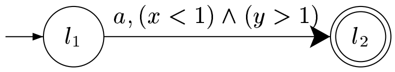

Fig. 4. Clock drift, however small, influences reachability

2.5 Choice of Clock Rates

An interesting generalization of timed automata is *rectangular automata* in which clocks increase at a rate that is bounded by constants [21]. Such a clock can be used to approximate a continuous variable. A rectangular automaton A over an alphabet Σ is a tuple $\langle V, V^0, V^F, X, E, low, high\rangle$, where the components V, V^0, V^F, X, and E are as in a timed automaton, and low and $high$ are functions from X to $\mathbb{Q}$. When time elapses each clock x increases at a rate bounded by $low(x)$ from below, and by $high(x)$ from above. The transition system S_A associated with the rectangular automaton A is defined as in case of timed automata. The only difference is in the transitions corresponding to elapse of time: for a state (s, ν), a time increment $\delta \in \mathbb{R}$, and a clock valuation μ, $(s, \nu) \xrightarrow{\delta} (s, \mu)$ holds if for each clock $x \in X$, there exists a rate $low(x) \leq r_x \leq high(x)$ such that $\mu(x) = \nu(x) + \delta \cdot r_x$.

Theorem 8. *The language accepted by a rectangular automaton is timed regular, and the language emptiness problem for rectangular automata is* PSPACE-*complete [21].*

The emptiness problem for rectangular automata is solved by translating rectangular automata to equivalent timed automata. Consider a rectangular automaton A. We obtain an equivalent automaton B as follows. For every clock x of A, B has two clocks: x_l whose rate is $low(x)$ and x_h whose rate is $high(x)$. We would like x_l and x_h to track the lower and upper bounds, respectively, on the possible values of the clock x whose rate can vary in the interval $[low(x), high(x)]$. Consider a switch of A with guard $x \leq c$. The corresponding switch in B has guard $x_l \leq c$, and update $x_h := c$. Analogously, the guard $x \geq d$ is replaced by the constraint $x_h \geq d$, with an accompanying adjustment $x_l := d$. This transformation preserves answers to reachability questions, and in fact, timed languages. The automaton B has clocks that have fixed rates, and can easily be transformed into a timed automaton simply by scaling. Note that in rectangular automata, a variable does not change its rate from one location to another, the enabling conditions compare variables with constants, and updates reset variables to constants. Relaxing any of these restrictions results in undecidability [21].

Rectangular automata are also useful to introduce “errors” in the clocks. For a timed automaton A, and a constant ε, let A^ε be the rectangular automaton obtained from A by setting $low(x) = 1 - \varepsilon$ and $high(x) = 1 + \varepsilon$ for all clocks

x. Thus, the clocks in A^ε have a drift bounded by ε. A location s of a timed automaton A is said to be *limit-reachable* if s is reachable in the perturbed automaton A^ε, for every $\varepsilon > 0$. Obviously, reachability implies limit reachability, but not vice versa [33]. For instance, the language of the automaton of Figure 4 is nonempty as long as the there is a non-zero drift for the two clocks. It is possible to compute the set of limit-reachable locations by modifying the search in the region automaton $R(A)$. For example, in Figure 4, in the initial location, the region $0 < x = y < 1$ is reachable. Since it touches the region $0 < x < y = 1$, which, in turn, touches the region $0 < x < 1 < y$, the latter is declared limit-reachable, and this makes the discrete switch to the final location possible. The computation, in general, requires identifying the so-called limit-cycles in the region graph [33].

Theorem 9. *Given a timed automaton A, the problem of deciding whether a location is limit reachable is* PSPACE*-complete [33].*

Instead of perturbing the clock rates, if we perturb the guards, that is, replace every $x \leq c$ by $x \leq c + \varepsilon$ and every $x \geq c$ by $x \geq c - \varepsilon$, and ask if a location is reachable for every positive perturbation ε of the guards, then the problem is solvable by similar techniques [17].

2.6 Weighted Automata and Optimal Reachability

A *weighted timed automaton* consists of a timed automaton A, a cost function J that maps every location and every switch to a nonnegative rational number. For a location $s \in V$, $J(s)$ is the cost of staying in s per unit time, and for a switch $e \in E$, $J(e)$ is the cost of a discrete switch corresponding to e. The cost function leads to costs on the edges of the underlying transition system S_A: the transitions of the form $(s, \nu) \xrightarrow{\delta} (s, \nu+\delta)$ have cost $\delta \cdot J(s)$, and transitions due to a switch e have cost $J(e)$. The *optimal reachability* problem for weighted timed automata is to determine the cost of the shortest path from an initial state to a final state, and thus, is a generalization of the classical shortest path problem in weighted automata. Formally, given a timed automaton A, and a cost function J, the optimal cost of reaching the final set V^F is the infimum over costs c such that there is a path of cost c from an initial state to a final state. The solution to this problem has been proposed in [7] (see also [8] for an alternative approach). Consider a path in the underlying graph of the timed automaton from an initial location to a final location. There can be many runs corresponding to the sequence of discrete switches specified by such a path, depending on the time spent between successive switches. However, since the constraints imposed by the resets and guards are linear, and so is the cost function, in an optimal run the times of switches will be at corner points (or arbitrarily close to corner points if the corner points are ruled out by the constraints).

In a more general version of the optimal reachability problem, we are given a source region (that is, some constraints on the initial values of the clocks), and we want to compute optimal costs for all the states in the source region.

It is possible to construct a weighted graph whose nodes are "refined" regions and edges are annotated with parametric costs that are linear functions of the clock values in the source state. The size of this graph, like the region graph, is exponential in the timed automaton. Fixing a source state determines the costs on all the edges, and optimal cost to reach any of the locations (or regions) can be computed in PSPACE (see also [12]). However, the number of parameters is same as the number of clocks, and if wish to compute a symbolic representation of the optimal cost to reach a target as a function of the source state, this approach gives a doubly exponential solution.

Theorem 10. *Given a timed automaton A, and a cost function J, the optimal cost of reaching a final state can be computed in* PSPACE.

3 Inclusion, Equivalence and Universality

3.1 Undecidability

The *universality* problem for timed automata is to decide, given a timed automaton A, whether A accepts all timed traces, i.e. whether $L(A) = T\Sigma^*$. For automata on discrete words, this is decidable as one can complement the automaton A and check for emptiness. This approach does not work for timed automata since, as we saw earlier, timed automata are not closed under complementation. In fact, it turns out that the problem is undecidable:

Theorem 11. *The universality problem for timed automata is undecidable [2].*

The proof proceeds by encoding computations of a 2-counter machine (or a Turing machine) using timed words where every unit time interval encodes a configuration of the machine. Copying between successive configurations is achieved by requiring that every event in one interval has a matching event distance 1 apart in the next interval. While this requirement cannot be captured by a timed automaton, the complement can be accepted by a nondeterministic timed automaton that guesses the errors (that is, events with no matches in the following interval).

The inclusion problem is to check, given two timed automata A and B, whether $L(A) \subseteq L(B)$. This is an interesting question from the formal methods perspective as it corresponds to the model-checking problem: given a system modeled using A and a specification modeled as B, is the set of behaviors of A contained in the the language defined by B?. The equivalence problem is to check, given A and B, whether $L(A) = L(B)$.

Since the set of all timed words is timed-regular, the universality problem reduces to both the inclusion and equivalence problems, and we have:

Corollary 1. *The inclusion and equivalence problems for timed automata are undecidable.*

Due to the interest in model-checking timed systems modeled as timed automata, there has been intense research over the years to find subclasses of timed automata for which the inclusion problem is decidable. We review some of them here.

3.2 Deterministic Timed Automata

A timed automaton A is *deterministic* if (1) V^0 contains only one location, (2) there are no switches labeled with ϵ, and (3) for every pair of distinct switches (s, a, g, λ, s') and $(s, a, g', \lambda', s'')$ with the same source location and label, the guards g and g' are disjoint (i.e. the sets of clock valuations that satisfy g and g' are disjoint). These requirements ensure that A has at most one run on a given timed word, and consequently, complementation can be achieved by complementing the set of final states. The properties of deterministic timed automata are summarized below:

Theorem 12. *Deterministic timed automata are closed under union, intersection, and complementation, but not under projection. The language emptiness, universality, inclusion, and equivalence problems for deterministic timed automata are* PSPACE*-complete [2].*

Unlike classical automata, deterministic timed automata are strictly less expressive than the nondeterministic ones, and in particular, the language of the automaton of Figure 1 cannot be specified using a deterministic timed automaton. Given a timed automaton A, the problem of checking whether there exists an equivalent deterministic timed automaton is not known to be decidable (see [35] for a discussion).

An interesting extension of deterministic timed automata is *bounded 2-way deterministic timed automata* [5]. Automata in this class deterministically traverse a timed word from left to right, but can stop and reverse direction to read the word backwards from that point. For example, consider the language consisting of all words of the form $(a, t_0)(a, t_1) \dots (a, t_k)(b, t')$ such that there exists some $i \leq k$ with $t' = t_i + 1$ (i.e. there is some a-event which is exactly one unit of time before the b-event). This language is not accepted by a (forward) deterministic automaton, but can be accepted by an automaton that goes to the end of the word, sets a clock to the time of the last event, and traverses the word backwards looking for the matching a event. For decidability, it is required that there exists a bound n such that any symbol of any word is read at most n times. Such a bounded timed automaton (even a nondeterministic one) can be simulated by a single-pass forward nondeterministic automaton as it simply needs to guess the positions where the clocks are reset on the bounded number of passes. In the deterministic case, the expressive power strictly increases with the bound n. These deterministic bounded two-way automata also preserve the crucial property that there is at most one run on each timed word, and consequently, they are closed under all boolean operations, and checking whether $L(A) \subseteq L(B)$, where A is a timed automaton and B is a bounded 2-way deterministic automaton, is decidable.

3.3 Digitization

An important subclass of timed automata for which the inclusion problem is decidable involves the notion of *digitization*. A timed language L is said to be closed under digitization if discretizing a timed word $w \in L$ by approximating the events in w to the closest tick of a discrete clock results in a word that is also in L.

Formally, for any $t \in \mathbb{R}$ and for any $0 \le \varepsilon \le 1$, let $[t]_\varepsilon$ be $\lfloor t \rfloor$, if $t < \lfloor t \rfloor + \varepsilon$, and $\lceil t \rceil$ otherwise. We extend this to timed words: if $w = (a_0, t_0), \ldots (a_k, t_k)$, then $[w]_\varepsilon = (a_0, [t_0]_\varepsilon) \ldots (a_k, [t_k]_\varepsilon)$. Intuitively, $[w]_\varepsilon$ is the word obtained when events are observed using a discrete clock with offset ε. For a timed language L, $[L]_\varepsilon = \{[w]_\varepsilon \mid w \in L\}$.

A timed language L is *closed under digitization* [22] if for every $w \in L$ and for every $\varepsilon \in [0, 1]$, $[w]_\varepsilon \in L$, i.e. if for every $\varepsilon \in [0, 1]$, $[L]_\varepsilon \subseteq L$. L is said to be *closed under inverse digitization* if it is the case that whenever u is a timed word such that for every $\varepsilon \in [0, 1]$, $[u]_\varepsilon \in L$, then u itself belongs to L.

For any timed language L, let $\mathbb{Z}(L)$ be the set of timed words in L in which every event happens at an integer time. Note the relation to the sampled semantics of Section 2.2: for a timed automaton A, $L^1(A) = \mathbb{Z}(L(A))$.

Lemma 1. *[22] Let L be closed under digitization and L' be closed under inverse digitization. Then $L \subseteq L'$ iff $\mathbb{Z}(L) \subseteq \mathbb{Z}(L')$.*

The proof of the above lemma runs as follows: Assume $\mathbb{Z}(L) \subseteq \mathbb{Z}(L')$. If $u \in L$, then $[u]_\varepsilon \in L$ for every $\varepsilon \in [0, 1]$ (since L is closed under digitization); hence $[u]_\varepsilon \in \mathbb{Z}(L) \subseteq \mathbb{Z}(L')$, for every $\varepsilon \in [0, 1]$, which in turn means that $u \in L'$ (since L' is closed under inverse digitization).

It is easy to see that timed languages over Σ in which events occur only at integral times are in one-to-one correspondence with untimed languages over $\Sigma \cup \{\surd\}$, where $\surd$ denotes the passage of one unit of time. For example, the trace $(a_0, 1)(a_1, 1)(a_2, 3)$ corresponds to the untimed word $\surd a_0 a_1 \surd\surd a_2$. For any timed word in which events occur at integral times, let $Tick(w)$ denote the corresponding word over $\Sigma \cup \{\surd\}$. Given a timed automaton A accepting L, we can effectively construct an automaton over $\Sigma \cup \{\surd\}$ accepting $Tick(\mathbb{Z}(L))$, using the region automaton for A. Hence, checking $\mathbb{Z}(L) \subseteq \mathbb{Z}(L')$ boils down to checking inclusion between two untimed languages, which is decidable. This gives:

Theorem 13. *[22] Given timed automata A and B, where $L(A)$ is closed under digitization and $L(B)$ is closed under inverse digitization, the problem of checking whether $L(A) \subseteq L(B)$ is decidable.*

Open timed automata are timed automata where all atomic clock constraints in guards are of the form $x < c$ or $x > c$, i.e. atomic guards of the form $x \le c$ and $x \ge c$ are disallowed. Similarly, *closed timed automata* are those in which all atomic guards are of the form $x \le c$ or $x \ge c$. The following is then true:

Proposition 1. *[22, 31] Closed timed automata are closed under digitization, and open timed automata are closed under inverse digitization.*

Corollary 2. *Given a closed timed automaton A and an open timed automaton B, the problem of checking if $L(A) \subseteq L(B)$ is decidable.*

Turning to the universality problem, since checking whether a timed automaton A accepts all timed words is the same as asking whether $T\Sigma^* \subseteq L(A)$, and since $T\Sigma^*$ is closed under digitization, it follows that:

Theorem 14. *[22, 31] The universality problem for open timed automata (or any class of timed automata that are closed under inverse digitization) is decidable.*

Note that the above crucially uses the fact that our definition of timed words allows several events to happen at the same time, i.e. the timed words are *weakly* monotonic. If this were disallowed and it was required that time strictly elapse between events, then we have as universe the set of all *strongly* monotonic words, which is not closed under digitization. It turns out that checking universality of open timed automata is undecidable in the domain of strongly monotonic words. Also, for closed timed automata, universality is undecidable regardless of whether the universe is weakly or strongly monotonic [31].

Note that open timed automata are defined syntactically by placing restrictions on the structure of the automata while closure under digitization is a semantic property of languages. Given automata A and B, one can always check whether $\mathbb{Z}(L(A)) \subseteq \mathbb{Z}(L(B))$. If we could decide whether A is closed under digitization and whether B is closed under inverse digitization, we would know whether we can use the above test to check language inclusion. It turns out that the former is decidable but the latter is not:

Theorem 15. *[31] Given a timed automaton A, checking whether $L(A)$ is closed under digitization is decidable, while the problem of checking whether $L(A)$ is closed under inverse digitization is undecidable (even if A is a closed timed automaton).*

3.4 Robust Timed Automata

Since the undecidability of universality and inclusion problems were shown using the fact that events that are precisely one unit (or an integral number of units) apart can be related, and hence used to count, this led to the belief that introducing some fuzziness in acceptance could alleviate the problem. Also, in practice, no timed system can be modeled and observed with such arbitrary accuracy a timed automaton provides.

The definition of robust timed automata addresses this. Given a timed automaton, under the robust semantics a word is accepted if and only if a dense subset "around" the word is accepted by the timed automaton. In this definition, a word that is accepted by the timed automaton may be rejected in the robust semantics if it is an isolated accepted trace, while a word that is rejected by the timed automaton can be accepted under the robust semantics if it is surrounded by a dense set of accepted traces.

Formally, let us first define a metric d on timed words. Let w and w' be two timed words. If $untime(w) \neq untime(w')$, then $d(w, w') = \infty$. Otherwise, if $w = (a_0, t_0) \dots (a_k, t_k)$ and $w' = (a_0, t'_0) \dots (a_k, t'_k)$, then $d(w, w') = \max\{|t_i - t'_i| \mid 0 \leq i \leq k\}$. In other words, the distance between two timed words (whose untimed components are identical) is the maximum difference in time between corresponding events in the two words. We refer to open and closed sets of timed words with regard to this metric.

The robust semantics can now be defined as follows. Given a timed automaton A accepting L, let L^c denote the smallest closed set containing L. Then the robust language accepted by the automaton, $L_R(A)$, is the interior of L^c, which is the largest open set contained within L^c.

In this subsection, to clearly distinguish between the standard semantics and the robust one, we refer to the former as *precise semantics*. In the original paper [20], the robust semantics of a timed automaton was defined as a collection of *tubes* as opposed to a collection of timed words. A tube is any set of timed words which is open (i.e. for each timed word in the tube, some ε-neighborhood should be contained in the tube). Here we adopt a slightly different semantics by defining the robust semantics to be the set of all timed words which belong to some tube that is robustly accepted.

The robust language of any timed automaton is, by definition, open. Also, it turns out that the (precise) languages accepted by open timed automata are also open. However, open timed automata and robust timed automata have incomparable expressive power (i.e. there are timed languages that are accepted by open timed automata which are not acceptable by robust timed automata and vice versa) [31].

Despite the involved definition of robust acceptance, emptiness for robust timed automata is decidable:

Theorem 16. *[20] The emptiness problem for robust timed automata is* PSPACE*-complete.*

The proof proceeds by showing that for any timed automaton A, we can construct an open timed automaton A^o such that both accept the same robust languages, i.e. $L_R(A) = L_R(A^o)$. Since the precise language of this open timed automaton is open, i.e. $L(A^o)$ is open, it follows that the robust language of A is nonempty iff the precise language of A^o is nonempty (i.e. $L_R(A) \neq \emptyset$ iff $L(A^o) \neq \emptyset$), which is decidable. One can in fact show that the untimed language corresponding to the robust language accepted by A is regular (as is true for the precise semantics).

However, it turns out that despite robustness, robust timed languages are not closed under complement (and hence not determinizable) [20, 26]. We give a new proof here. Consider the timed automaton A depicted in Figure 5, which accepts (in the precise semantics) the language L consisting of all timed words w such

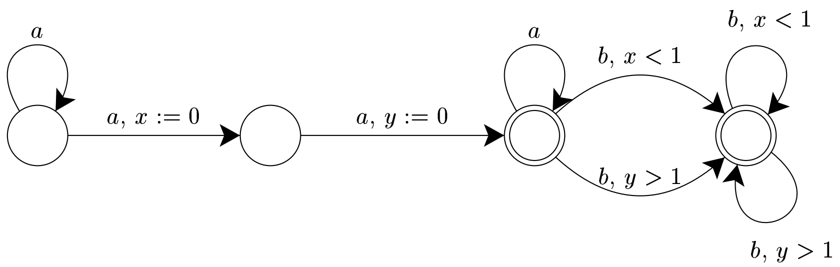

Fig. 5. A noncomplementable robust automaton

that the untimed word of w is in a^*b^* and there are two consecutive a-events at times t and t' such that there are no b-events in the range $[t+1, t'+1]$. It is easy to see that the robust language accepted by A is also L.

The robust complement of A, denoted by $\overline{L}$, consists of the set of all words w such that either the untimed word of w is not in a^*b^* or for every two consecutive a-events, there is at least one b-event in the *open* range $(t+1, t'+1)$. We show that $\overline{L}$ is not robustly acceptable by any timed automaton. We claim that a word is in the untimed language of $\overline{L}$ iff it is in $\Sigma^*.b.\Sigma^*.a.\Sigma^*$ or is of the form $a^m b^n$ where $n \geq m-1$. This claim will show that $\overline{L}$ cannot be robustly accepted, since $untime(\overline{L})$ is non-regular. The only interesting part is to show that there is no word whose untimed word is $a^m b^n$ with $n < m-1$. Assume there is such a word τ. By robust acceptance, we can find a word τ' close to τ whose untimed word is the same as that of τ but where all a-events occur at different times. Then, it is easy to see that the a-events define $m-1$ intervals which cannot be filled by n b-events and hence τ' is not in the language, which is a contradiction.

The above mechanism of sandwiching a related event in an interval one unit away from a pair of consecutive events, gives a way to maintain counters. A similar mechanism is used to encode configurations of a Turing machine in [24], where the authors show that a robust timed automaton can accept all wrong configuration sequences of a Turing machine, making universality of robust timed automata undecidable.

Turning to the notions defined in the last subsection, the languages defined by robust automata are closed under inverse digitization [31]. However, unlike regular timed languages, checking whether the robust language of a timed automaton is closed under digitization is undecidable [31].

Also, in sharp contrast to the precise semantics of timed automata, it turns out that the discrete-time language accepted by robust timed automata need not be regular. That is, there are robust timed automata A such that $\mathbb{Z}(L_R(A))$ is not regular. Consequently, there are timed languages that can be accepted by timed automata under the robust semantics which cannot be accepted by timed automata under the precise semantics (and vice versa).

The nonregularity of $\mathbb{Z}(L_R(A))$ seems to render digitization techniques inapplicable for checking inclusion of robust timed automata. In fact, the decidability status of the integral language emptiness under the robust semantics (i.e. given an automaton A, to check whether $\mathbb{Z}(L_R(A)) \neq \emptyset$) is not known. Also, introducing imprecision using infinitesimal clock drift (recall the definition of A^ε from Section 2.4) as a way of defining semantics, and its relationship to the robust semantics has not been studied.

3.5 Restricting Resources

One approach to get a more tractable subclass is to restrict the resources a timed automaton can use. The original proof showing that inclusion of timed automata is undecidable also showed that timed automata with two clocks already renders the inclusion problem undecidable [2].

For timed automata with one clock, however, a recent result shows that checking inclusion (i.e. checking if $L(A) \subseteq L(B)$) is decidable when B has only one clock [32]. The proof is based on techniques used to solve problems on *infinite* graphs akin to those used to solve problems involving coverability in Petri nets.

The paper [32] also shows that the problem of checking whether $L(A) \subseteq L(B)$ is decidable if the only constant that appears in the guards of B is 0. The proof goes by showing that B can be determinized. The essence of the idea is this: Consider the region automaton for B. The only information we need to maintain is whether each clock is 0 or greater than 0—the ordering of fractional parts of clocks need not be recorded as any region has at most one timed successor (the one with every clock greater than 0). Using now a clock, we can simulate a subset construction on the region automaton and turn it into a timed automaton where the clock is reset on every event and is used to check whether any amount of time has elapsed since the last event.

Theorem 17. *[32] The problem of checking, given two timed automata A and B, whether $L(A) \subseteq L(B)$, is decidable if B does not have any ϵ-labeled switches and either:*

- *B uses only one clock, or*
- *B uses guards involving the constant 0 only.*

The above results are the only known decidability results in this category. In fact, the following relaxations of these restrictions on a given automaton A, renders the universality problem undecidable [32]:

- A has two clocks and a one-event alphabet, or
- A has two clocks and uses a single non-zero constant in the guards, or
- A has a single location and a one-event alphabet, or
- A has a single location and uses a single non-zero constant in the guards.

3.6 Event Clock Automata

The essential power of nondeterminism in timed automata lies in its ability to reset clocks nondeterministically, as will become clear later in this subsection. The class of *event-recording automata* [4] are timed automata with a fixed set of clocks, a clock x_a for each $a \in \Sigma$, where x_a gets reset every time a occurs. There are no ϵ-labeled switches. Event-recording automata thus have switches labeled (a, g) instead of (a, g, λ), as it is implicitly assumed that $\lambda = \{x_a\}$.

An event-recording automaton A can be easily determinized. First, we can transform A to an automaton B such that if G is the set of guards used on the transitions, then G is "minimal" in the sense that for any two guards g and g' in G, there is no clock valuation that satisfies both g and g'. Then, we can do a subset construction on this automaton. Let $B = \langle V, V^0, V^F, X, E\rangle$. Then, we can build a deterministic event recording automaton $C = \langle 2^V, \{V^0\}, F, X, E'\rangle$ where for any $S \subseteq V$, $a \in \Sigma$, $g \in G$, $(S, a, g, S') \in E'$ where $S' = \{v' \in V \mid \exists v \in S.(v, a, g, v') \in E\}$. The set F contains the sets $S \subseteq V$ such that $S \cap V^F \neq \emptyset$. It is easy to see that C is deterministic and accepts the same language as B does. Note that a similar construction fails for timed automata since for a set S, there could be two states $v, v' \in S$ with edges (v, g, λ, v_1) and (v', g, λ', v_1'), where $\lambda \neq \lambda'$.

An event-recording automaton at any point on the input word has access to a clock x_a, for each $a \in \Sigma$, whose value is the time that has elapsed since the last a-event. *Event clock automata* are an extension in which the automaton also has access to a *prophecy* clock y_a (for each $a \in \Sigma$) whose value at any point is the time that must elapse before the next a-event happens. For, example, in the timed word $(a, 0.4)(b, 0.5)(a, 0.7)(b, 0, 9)(a, 0.95)$, when reading the third event in the word, the clock $x_a = 0.3$ and $y_a = 0.25$.

Observe that prophecy clocks add to the expressiveness: the language of timed words such that the untimed word is in a^*b and there is some a event one time unit before b, is not accepted by any event recording automaton, or even any deterministic timed automaton, but can easily be accepted by an event clock automaton. For every event clock automaton, we can construct a (nondeterministic) timed automaton that accepts the same language. Event-recording clocks x_a do not cause any problem, of course, as we can reset the clock x_a at each a-event. To handle prophecy clocks is more tricky. The timed automaton simulates the event-clock automaton, and if at an event a guard demands $y_b < c$, then we can take the action and postpone the checking of this constraint. We do this by resetting a new clock $z_{y_b<c}$ and check at the next b-event that $z_{y_b<c} < c$ holds. If we meet another transition before the next b-event which also demands $y_b < c$ hold, then we can ignore it as checking $y_b < c$ at an earlier position is a stronger condition. Similarly, constraints of the form $y_b > c$ can be handled. Note that the resulting automaton can be nondeterministic as multiple edges that demand different constraints on the prophecy clocks can be enabled.

Since the values of any clock of an event clock automaton at any time depends only on the word w (and not on the run of the automaton), it turns out that event-clock automata can be complemented. Let A be an event clock automaton and let the guard constraints G used in A be "minimal". Also, let us assume that the guards of switches with identical source location and identical label cover the set of all clock valuations so that some guard is always enabled. Let Π be the set of all (a, g) where $a \in \Sigma$ and $g \in G$. Note that the transitions of A are labeled using symbols in Π and that Π is finite.

Consider words in Π^*. For any word $\pi \in \Pi^*$, we can associate a set of timed words $tw(\pi)$ corresponding to it. Formally, if $\pi = (a_0, g_0) \ldots (a_n, g_n)$, then $tw(\pi)$ contains the set of all timed words of the form $(a_0, t_0) \ldots (a_n, t_n)$ where, for any $i \leq n$, the set of event-recording and prophecy clocks at (a_i, t_i) satisfy the guard g_i.

In fact, if we denote the set of *symbolic* words accepted by A as $L_{sym}(A)$ (which is a regular subset of Π^*), it is easy to see that $L(A) = \bigcup_{\pi \in L_{sym}(A)} tw(\pi)$ [18].

Notice that for any timed word w, there is a word $\pi \in \Pi^*$ such that $w \in tw(\pi)$. In fact, this symbolic word is unique, by the minimality of the guards. Consequently, the timed words corresponding to words in $\Pi^* \setminus L_{sym}(A)$ form the complement of $L(A)$, i.e. $tw(\overline{L_{sym}(A)}) = \overline{L(A)}$. Hence we can complement the event clock automaton A by constructing an automaton A' accepting the complement of $L_{sym}(A)$ and by viewing A' as an event clock automaton. We can indeed even build a deterministic automaton for $L_{sym}(A)$ and by viewing it as an event-clock automaton we would get a *deterministic* event clock automaton equivalent to A. For event-recording automata A, this construction in fact yields a deterministic timed automaton equivalent to A.

We have the following results:

Theorem 18. *[4] Event clock automata are effectively closed under complementation. Further, given a timed automaton A and an event clock automaton B, the problem of checking whether $L(A) \subseteq L(B)$ is* PSPACE*-complete.*

Choosing recording clocks x_a and prophecy clocks y_a, for every symbol $a \in \Sigma$, is rather arbitrary, and one can generalize the notion of events with the corresponding recording and predicting clocks. For example, the occurrence of two a's exactly one unit of time apart can be an event for which we may want to keep recording and prophecy clocks. The property we would like to maintain is that the events are *determined* by the word, and not by a particular run of an automaton on the word.

The class of *recursive event clock automata* [25] are defined using this principle. These automata consist of a finite collection of automata, one at each level $\{1, \ldots, k\}$. The automaton at each level A_i uses events that are defined by the automaton at level A_{i-1} (A_1 is a simple event clock automaton). The notion of events is complex: essentially each automaton A_i comes as a pair of event clock automata (A_i^l, A_i^r) and an event is generated by A_i at time t if the prefix of the word till time t is accepted by A_i^l and the suffix from time t is accepted by A_i^r. The automaton at level i then uses clocks of the form x_j and y_j, $(j < i)$, where x_j and y_j are recording and prophecy clocks for events defined by the automaton A_j. The main result is that checking if $L(A) \subseteq L(B)$ is decidable, when A is a timed automaton and B is a recursive event-clock automaton. The class of languages defined by recursive event-clock automata has logical characterizations using real-time temporal logics [25, 34, 18], but its expressive power

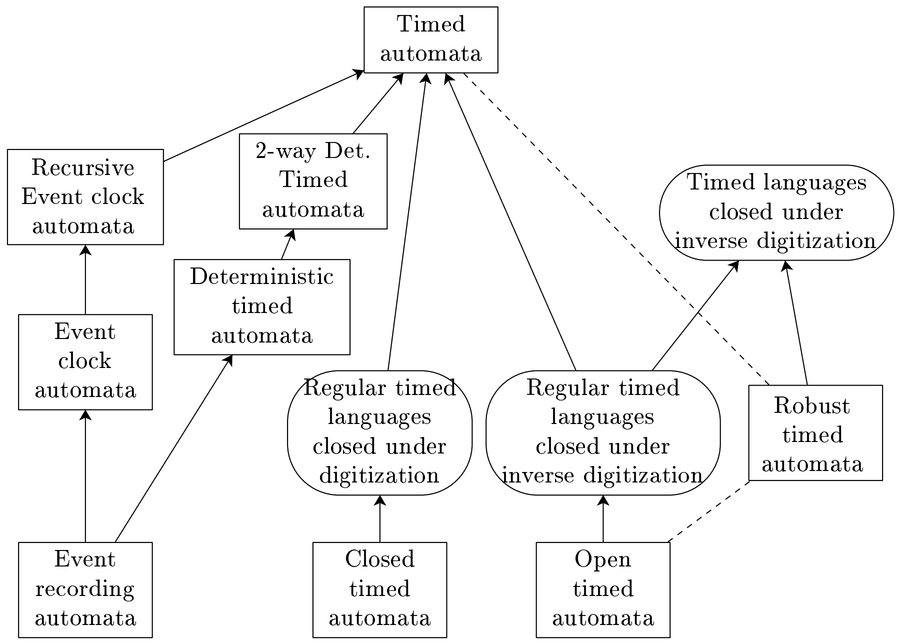

Fig. 6. The various classes of timed languages.
An arrow from C to D denotes that the class defined by C is a subclass of that defined by D. Dotted lines emphasize that certain classes are not comparable.

with respect to deterministic bounded two-way automata has not been studied. The relationship among various classes is summarized in Figure 6.

3.7 Resource-bounded Inclusion

We present in this section a result that shows that checking whether a timed automaton with limited resources can exhibit an evidence to the fact that $L(A)$ is not a subset of $L(B)$, is decidable. This result is derived from ideas in [1, 19].

The *resources* of a timed automaton are the following: the number of clocks that it uses, the granularity $1/m$ with which it makes observations of the clocks, and the maximum constant it uses. The maximum constant, however, is not important, as for any timed automaton A, there exists an equivalent timed automaton B with ϵ-transitions which uses the same number of clocks, has the same granularity as A, but with maximum constant 1 in the guards. We can construct B such that it simulates A, except that it keeps track of $\lfloor x \rfloor$, for each clock x, in its control state, and uses the clock only to keep track of $x - \lfloor x \rfloor$.

The number of clocks and the granularity of observation are however important—increasing the number of clocks or decreasing the granularity from say $1/m$ to $1/2m$ strictly increases the class of languages a timed automaton can accept.

Given timed automata A and B, and resources $(k, 1/m)$, we now want to know whether there is an automaton C with granularity $(k, 1/m)$ which can be an evidence to the fact that $L(A)$ is not contained in $L(B)$. More precisely, is there such a C such that $L(A) \cap L(C) \neq \emptyset$ but $L(B) \cap L(C) = \emptyset$? We show that this is a decidable problem.

Let us fix resources $(k, 1/m)$. Let $X_k = \{x_1, \ldots, x_k\}$ be a set of k-clocks and let $G_{1/m}$ denote the set of all minimal guards formed using boolean combinations of constraints of the form $x_i \leq 1/m$ and $x_i < 1/m$, where $x_i \in X_k$. Let $\Pi = \{(a, g, \lambda) \mid a \in \Sigma', g \in G_{1/m}, \lambda \subseteq X_k\}$. Note that for any timed automaton C which has minimal guards on transitions, the symbolic language it accepts is a subset of Π^*.

Each word $\pi \in \Pi^*$ defines a set of timed words $tw(\pi)$ over Σ which is basically the set of timed words that would be accepted by a timed automaton along a run that is labeled with π. The question of the existence of a C that witnesses that $L(A)$ is not a subset of $L(B)$ boils down to finding whether there is some symbolic word $\pi \in \Pi^*$ such that $tw(\pi) \cap L(A) \neq \emptyset$ and $tw(\pi) \cap L(B) = \emptyset$.

The following lemma will help capture the set of all such witnesses:

Lemma 2. *[19] Let D be any timed automaton over Σ and let Π be a symbolic alphabet for granularity $(k, 1/m)$ as above. Then, the set of all $\pi \in \Pi^*$ such that $tw(\pi) \cap tw(D) \neq \emptyset$ is regular.*

The proof follows using the intersection construction for timed automata. Let E be the automaton accepting Π^*. Essentially, the automaton we are looking for is the region automaton accepting the product of D and E. When we take a product transition, however, we label this transition with the Π-label that was involved in the transition.

Consequently, R_A, the set of all words π in Π^* such that $tw(\pi) \cap L(A) \neq \emptyset$ is regular, and the set R_B of all words π in Π^* such that $tw(\pi) \cap L(B) = \emptyset$, is also regular. We can now check whether $R_A \cap R_B$ is empty, which is decidable, and we have:

Theorem 19. *Given timed automata A and B, and a resource constraint $(k, 1/m)$, the problem of checking whether there is an automaton C with granularity $(k, 1/m)$ such that $L(A) \cap L(C) \neq \emptyset$ and $L(B) \cap L(C) = \emptyset$ is decidable.*

4 Discussion

This survey attempts to collect, unify, and explain selected results concerning reachability and language inclusion for timed automata and its variants. The theoretical questions studied in the literature, but not addressed in this survey, include timed ω-languages, connections to monadic logics, regular expressions, and circuits, branching-time equivalences such as timed (bi)simulations, model checking of real-time temporal logics, analysis of parametric timed automata, and games and controller synthesis.

The reachability problem is the most relevant problem in the context of formal verification, and its complexity class is PSPACE. A large number of heuristics have been proposed to efficiently implement the reachability algorithm. All these involve searching the region automaton, either explicitly, or using symbolic encoding of regions using zones (see $[6, 29, 16, 36, 11]$ for sample tools). Many of these optimizations have been devised so as to avoid enumerating all possible numerical combinations of the (integral) clock values. We believe that new insights can be obtained by exploring the following theoretical question [27]. Consider the special case when the graph formed by locations and edges of a timed automaton A is acyclic. Even in this case, the region automaton can be exponential, and the shortest path to a target region can be of exponential length. However, it is easy to see that the problem is in NP: the number of discrete switches along the path to the target is linear, it suffices to guess the regions when these discrete switches occur, and it is easy to verify the feasibility of the guess. The problem can also be shown to be NP-hard. The NP upper bound also holds if we allow a single self-loop switch on each location. We conjecture that this bound continues to hold when the strongly connected components in the graph are small: if the number of edges in each strongly-connected component of the graph formed by the locations and edges of a timed automaton is bounded, then the reachability problem is in NP.

The fact that the language "some two a symbols are distance 1 apart" is timed regular has led to the belief that timed automata are too powerful in terms of precision and unbounded nondeterminism, causing noncomplementability and undecidable language inclusion problem. The various solutions such as event clock automata, robust automata, open timed automata, have been proposed to address this issue. However, no solution has emerged as a convincing alternative, and research in obtaining a class of automata with properties more attractive than those of timed automata continues. We believe that introducing a small drift in the clocks of timed automata is a natural and simple way to introduce imprecision. Let us call a timed regular language L to be a *perturbed* language if there exists a timed automaton A and an error $\varepsilon > 0$ such that $L = L(A^{\varepsilon})$. We conjecture that the class of perturbed languages has a decidable language inclusion problem.

Acknowledgments We thank Patricia Bouyer, Deepak D'Souza, Tom Henzinger, Joel Ouaknine and Jean-Francois Raskin for useful comments on the draft of this manuscript.

References

1. R. Alur, C. Courcoubetis, and T. Henzinger. The observational power of clocks. In *CONCUR '94: Fifth International Conference on Concurrency Theory*, LNCS 836, pages 162–177. Springer-Verlag, 1994.
2. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

3. R. Alur, T. Feder, and T. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.
4. R. Alur, L. Fix, and T. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211:253–273, 1999. A preliminary version appears in *Proc. CAV'94*, LNCS 818, pp. 1–13.
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 and R. Kurshan. Timing analysis in COSPAN. In *Hybrid Systems III: Control and Verification*, LNCS 1066, pages 220–231. Springer-Verlag, 1996.
7. R. Alur, S. La Torre, and G. Pappas. Optimal paths in weighted timed automata. In *Hybrid Systems: Computation and Control, Fourth International Workshop*, LNCS 2034, pages 49–62, 2001.
8. G. Behrman, T. Hune, A. Fehnker, K. Larsen, P. Petersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In *Hybrid Systems: Computation and Control, Fourth International Workshop*, LNCS 2034, pages 147–161, 2001.
9. B. Bérard and C. Dufourd. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1–2):1–7, 2000.
10. B. Berard, P. Gastin, and A. Petit. On the power of non-obervable actions in timed automata. In *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS 1046, pages 257–268, 1996.
11. P. Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.
12. P. Bouyer, E. Brinksma, and K. Larsen. Staying alive as cheaply as possible. In *Proc. 7th Int. Workshop on Hybrid Systems: Computation and Control (HSCC 2004)*, LNCS 2993, pages 203–218. Springer, 2004.
13. P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable? In *Computer Aided Verification, 14th International Conference*, LNCS 2404, pages 464–479, 2000.
14. F. Cassez, T. Henzinger, and J. Raskin. A comparison of control problems for timed and hybrid systems. In *Hybrid Systems: Computation and Control, Fifth International Workshop*, LNCS 2289, pages 134–148, 2002.
15. C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In *Proceedings of the Third Workshop on Computer-Aided Verification*, LNCS 575, pages 399–409. Springer-Verlag, 1991.
16. 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.
17. M. De Wulf, L. Doyen, N. Markey, and J. Raskin. Robustness and implementability of timed automata. In *Proc. FORMATS*, 2004.
18. D. D'Souza. A logical characterisation of event recording automata. In *Proc. 6th Int. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'00)*, LNCS 1926, pages 240–251. Springer, 2000.
19. D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *Proceedings of the 19th Symposium on Theoretical Aspects of Computer Science*, LNCS 2285, pages 571–582. Springer, 2002.
20. V. Gupta, T. Henzinger, and R. Jagadeesan. Robust timed automata. In *Hybrid and Real Time Systems: International Workshop (HART'97)*, LNCS 1201, pages 48–62. Springer, 1997.

21. T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata. *Journal of Computer and System Sciences*, 57:94–124, 1998.
22. T. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *ICALP 92: Automata, Languages, and Programming*, LNCS 623, pages 545–558. Springer-Verlag, 1992.
23. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.
24. T. Henzinger and J. Raskin. Robust undecidability of timed and hybrid systems. In *Hybrid Systems: Computation and Control, Third International Workshop*, LNCS 1790, pages 145–159, 2000.
25. T. Henzinger, J. Raskin, and P. Schobbens. The regular real-time languages. In *ICALP'98: Automata, Languages, and Programming*, LNCS 1443, pages 580–593. 1998.
26. P. Herrmann. Timed automata and recognizability. *Information Processing Letters*, 65(6):313–318, 1998.
27. S. La Torre, S. Mukhopadhyay, and R. Alur. Subclasses of timed automata with NP-complete reachability problem. Technical report, 2003. Unpublished.
28. F. Laroussinie, N. Markey, and P. Schnoebelen. Model checking timed automata with one or two clocks. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR 2004)*. Springer, 2004.
29. K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1, 1997.
30. J. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control (HSCC 2000)*, LNCS 1790, pages 296–309. Springer, 2000.
31. J. Ouaknine and J. Worrell. Revisiting digitization, robustness, and decidability for timed automata. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science*, 2003.
32. J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science*, 2004.
33. A. Puri. Dynamical properties of timed automata. In *Proceedings of the 5th International Symposium on Formal Techniques in Real Time and Fault Tolerant Systems*, LNCS 1486, pages 210–227, 1998.
34. J. Raskin and P. Schobbens. The logic of event clocks – decidability, complexity, and expressiveness. *Journal of Automata, Languages, and Combinatorics*, 4(3):247–286, 1999.
35. S. Tripakis. Folk theorems on determinization and minimization of timed automata. In *Proc. FORMATS*, 2003.
36. F. Wang. Efficient data structures for fully symbolic verification of real-time software systems. In *TACAS '00: Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Software*, LNCS 1785, pages 157–171, 2000.