Automata For Modeling Real-Time Systems

Rajeev Alur[1] David Dill[2]
Department of Computer Science
Stanford University, U.S.A.

ABSTRACT. To model the behavior of finite-state asynchronous real-time systems we propose the notion of *timed Büchi automata* (TBA). TBAs are Büchi automata coupled with a mechanism to express constant bounds on the timing delays between system events. These automata accept languages of *timed traces*, traces in which each event has an associated real-valued time of occurrence.

We show that the class of languages accepted by TBAs is closed under the operations of union, intersection and projections, and the trace language obtained by projecting the language accepted by a TBA is ω-regular. It turns out that TBAs are not closed under complement, and it is undecidable whether the language of one automaton is a subset of the language of another. This result is an obstruction to automatic verification. However, we show that a significant (proper) subclass represented by *deterministic timed Muller automata* (DTMA) is closed under all the boolean operations. Consequently, a system modeled by a TBA can be automatically verified with respect to a specification given as a DTMA.

1 Introduction

Modal logics and ω-automata for *qualitative* temporal reasoning about concurrent systems have been studied in great detail (selected references: [Ch74, WVS83, CES86, Pn86, Di88, CDK89]). These formalisms abstract away from time, retaining only the sequencing of events in a system. In the automata-theoretic approach, a system is modeled as a finite-state non-deterministic automaton on infinite strings (e.g. Büchi automaton); the language accepted by the automaton corresponds to the set of possible behaviors of the system. The operations useful for describing complex systems can then be viewed as language-theoretic operations. For example, parallel composition can be modeled using projection and intersection. Furthermore, the verification problem — whether every possible behavior of an implementation is a member of the set of behaviors allowed by the specification, now becomes the language inclusion problem. For Büchi automata there are known effective constructions for intersection and complementation. Also the language inclusion problem is decidable.

For a large class of systems - *real-time* systems, the system has to meet certain *hard real-time constraints* (e.g. "the system should respond within 2 seconds"), and the correctness of the system depends on the actual values of the delays. For the analysis of such systems we need to develop formalisms for *quantitative temporal reasoning.* System designers have realized the need for such models, and several ways to extend the existing formalisms have been suggested. Most of these attempts are tailored to specific applications and do not handle concurrency in a general way. Also, this work is generally somewhat *ad hoc*: questions such as formal semantics, soundness, expressiveness, and

[1]Supported by the NSF grant CCR-8812595, and the DARPA contract N00039-84-C-0211, and by the USAF office of Scientific Research under contracts 88-0281 and 90-0057.
[2]Supported by the NSF grant MIP-8858807.

closure properties have not been addressed. This problem has received relatively little attention from the theoreticians, though recently researchers have studied quantitative temporal logics ([JM86, Ko89, AH90, ACD90, Le90]).

The first question to address is how to incorporate time explicitly in the underlying formal semantics for processes. Most of the previous work on modeling real-time systems has used two basic approaches. *Discrete-time* models use the domain of integers to model time. This approach requires that continuous time be approximated by choosing some fixed quantum *a priori*, which limits the accuracy with which the system can be modeled. The *fictitious clock* approach introduces a special *tick* transition in the model. Here time is viewed as a global state variable that ranges over the domain of natural numbers, and is incremented by one with every *tick* transition (e.g. [AK83, AH90]). This model allows arbitrarily many transitions of any process between two successive *tick* transitions. The timing delay between two events is measured by counting the number of *tick*s between them. Consequently, it is impossible to state precisely certain simple requirements on the delays such as "the delay between two transitions equals 2 seconds." Both these models are used, despite their apparent inaccuracy, because they are straightforward extensions of existing temporal models.

In this paper we use a different model. We choose a dense domain to model time, and extend the trace semantics by associating with each event its time of occurrence. This allows an unbounded number of environment events between any two events of a system, and also allows a more accurate modeling of the timing delays in asynchronous systems.

To model finite-state real-time systems we define *timed automata.* A timed automaton is an ω-automaton with an auxiliary finite set of *clocks* which record the passage of time. The clocks can be reset with any state-transition of the automaton. The timing constraints are expressed by associating with the transitions enabling conditions which compare clock values with time constants. When coupled with acceptance criteria, such as Büchi acceptance, timed automata accept *timed traces,* sequences in which every element has an associated real-valued time.

Our work is based on the recent proposals by Dill ([Di89]) and Lewis ([Le89]) for extending state-graphs with timing constraints based on a continuous model of time. These papers provide a way of using the timing assumptions to verify qualititative temporal requirements. Later works use these models as interpretations for the formulas of branching-time real-time logics, and give model-checking algorithms ([ACD90, Le90]).

In this paper we use timed automata to define languages of (linear) timed traces, and study their language-theoretic properties. We show that the class of languages accepted by timed Büchi automata (TBA) is closed under intersection and projections. Consequently, there are effective constructions for operations such as parallel composition, hiding, and renaming for processes modeled as sets of timed traces in this class.

Given a TBA we show how to construct a Büchi automaton which accepts exactly those traces to which times can be attached consistently with the timing constraints of the given automaton. This gives an algorithm for deciding the emptiness of its language, and allows using the information about the timing delays to verify qualitative temporal requirements. However, the language inclusion problem is undecidable, which poses problems for automatic verification of real-time requirements. We define the notion of *determinism* for timed automata, and show that deterministic timed automata can be complemented. In particular, we show that deterministic timed Muller automata

(DTMA) are closed under all the boolean operations (but not under projections). Consequently, automatic verification is possible when a specification is expressed as a DTMA and an implementation is modeled using TBAs.

Outline: In section 2 we develop the essential aspects of semantics of timed traces for concurrent systems. In section 3 we define timed automata and discuss their closure properties and decision problems. In section 4 we present the complementable subclass of DTMAs, and show how it can be used for verification.

2 Timed trace semantics

To define the semantics of real-time processes, we first define a simple trace semantics for concurrent processes, and then introduce explicit time in the model.

2.1 Trace semantics for untimed processes

In trace semantics, we associate a set of observable *events* with each process (an example event might be an assignment of a value to a variable, or arrival of a message), and model the process by the set of all its *traces*. A trace is a (linear) sequence of events that may be observed when the process runs. All events are assumed to occur instantaneously. Hoare originally proposed such a model for CSP ([Ho85]). In our model we allow several events to happen simultaneously, and consider both finite and infinite traces.

Formally, given a set A of events, a *trace* is a finite or infinite word over $\mathcal{P}^+(A)$, the set of all nonempty subsets of A. An *untimed process* is a pair (A, X), where A is the set of its observable events and $X \subseteq \mathcal{P}^+(A)^\infty$ is the set of its possible traces. (For any alphabet Σ, Σ^∞ denotes the set of all finite and infinite strings over it.)

The *projection* of $\rho \in \mathcal{P}^+(A)^\infty$ onto $A_1 \subseteq A$ (written $\rho\lceil_{A_1}$) is formed by intersecting each event set in ρ with A_1 and deleting all the empty sets from the sequence. Note that the projection can be finite even if the original trace is infinite.

The following operations are helpful for constructing complex systems from the simpler ones :

- **Parallel Composition** : Given $P_1 : (A_1, X_1)$ and $P_2 : (A_2, X_2)$, their composition $P_1 \parallel P_2$ is a process with the event set $A_1 \cup A_2$ and trace set $\{\rho \in \mathcal{P}^+(A_1 \cup A_2)^\infty : \rho\lceil_{A_1} \in X_1 \wedge \rho\lceil_{A_2} \in X_2\}$).
- **Hiding** : Given a process $P : (A, X)$ and an event $a \in A$, $P \backslash a$ is the process $(A - \{a\}, \{\rho\lceil_{(A-\{a\})} : \rho \in X\})$.
- **Event renaming** : Given event sets A and A', a process $P : (A, X)$, and a bijective renaming map $\mu : A \to A'$, $\mu(P)$ is $(A', \{\mu(\rho) : \rho \in X\})$ (where μ is naturally extended to event sets, and then extended pointwise to finite and infinite sequences).

2.2 Timed traces

An untimed process models the sequencing of events but not the actual times at which the events occur. Timing can be added to a trace by coupling it with a sequence of real-valued times. We are interested only in the delays between the successive events, but, for convenience, we adopt the convention of assigning time 0 to the first event of a trace. We will use $\mathbf{R}$ to denote the set of nonnegative real numbers. The results of this paper still apply if we choose any other dense linear order, instead of $\mathbf{R}$, to model time.

A *time sequence* τ is a finite or infinite sequence over $\mathbf{R}$ satisfying the following constraints:

- *Initiality:* τ begins at time 0; that is, $\tau(0) = 0$.
- *Monotonicity:* τ increases strictly monotonically; that is, for all $i \geq 0$, $\tau(i) < \tau(i+1)$.
- *Progress:* For every $t \in \mathbf{R}$, there is some i such that $\tau(i) > t$.

The progress condition implies that only a finite number of events can happen in a bounded interval of time.

A *timed trace* over an alphabet Σ is a pair (ρ, τ) where $\rho \in \Sigma^\infty$, τ is a time sequence, and ρ and τ are of equal length (infinite sequences are considered to have length ω). A *timed language* over Σ is a set of timed traces over Σ. A *timed process* is a pair $P : (A, L)$ where L is a timed language over $\mathcal{P}^+(A)$.

The operations on untimed processes are extended in the obvious way to timed processes: the projection of a timed trace deletes events from ρ and leaves τ unchanged, unless the element in ρ becomes empty, in which case that element and the corresponding time are deleted. The definitions of parallel composition, hiding, and renaming are as before, except that they use the projection for timed traces. The times associated with events can be discarded by the *Untime* operation: $\mathit{Untime}[(A, L)]$ is defined to be $(A, \{\rho \in \Sigma^\infty : \exists \text{ time sequence } \tau \text{ such that } (\rho, \tau) \in L\})$.

Note that, in general, $\mathit{Untime}(P_1 \parallel P_2) \subseteq \mathit{Untime}(P_1) \parallel \mathit{Untime}(P_2)$ (they are not necessarily equal). In other words, the timing information retained with the traces constrains the set of possible traces when two processes are composed (which is why they are interesting).

3 Timed automata

Processes whose behaviors are regular are especially interesting because they can be represented using finite automata. The operations on processes can be implemented using standard constructions on the automata, and the inclusion problem is decidable. This forms a basis for automatic verification of finite-state concurrent systems, such as certain hardware devices and communication protocols.

3.1 ω-automata and untimed processes

An ω-automaton is essentially the same as a non-deterministic finite-state automaton, but with the accepting condition modified suitably so as to handle infinite input words also. These automata provide a finite representation for regular trace sets. Various types of ω-automata have been studied in the literature ([Bü62, Mc66, Ch74]). In this paper we will consider two types of ω-automata: Büchi automata and Muller automata.

We consider ω-automata with ε-transitions, which accept both finite and infinite traces; this is non-standard, but all of the basic results for these automata continue to hold (as do the constructions, with small modifications).

An *ω-automaton* M is a tuple $\langle \Sigma, \mathrm{S}, \mathrm{S}_0, \mathrm{E} \rangle$, where Σ is the input alphabet, S is a finite set of automaton states, $\mathrm{S}_0 \subseteq \mathrm{S}$ is a set of start states, and $\mathrm{E} \subseteq \mathrm{S} \times \mathrm{S} \times [\Sigma \cup \{\varepsilon\}]$ is a set of edges. If $\langle s, s', \sigma \rangle \in$ E then the automaton can change its state from s to s' reading the input symbol σ.

Given $\rho \in \Sigma^{\infty}$, we say that $r : s_0 \xrightarrow{\rho'_0} s_1 \xrightarrow{\rho'_1} s_2 \xrightarrow{\rho'_2} \ldots$ is a *run* of M over ρ, provided

- the input word read is ρ: ρ' is an infinite sequence over $[\Sigma \cup \{\varepsilon\}]$ such that ρ is ρ' with all ε's removed, and
- the run satisfies proper consecution requirements: for each $i \geq 0$, there is an edge in E from state s_i to s_{i+1} with label ρ'_i.

Furthermore, r is called an *initialized run* if s_0 is a start state.

A *Büchi automaton* M is an ω-automaton with an additional set $\text{F} \subseteq \text{S}$ of accepting states. A run r of M over a word $\rho \in \Sigma^{\infty}$ is an *accepting run* if for infinitely many i's, $s_i \in$ F. The language L(M) accepted by M consists of the traces $\rho \in \Sigma^{\infty}$ such that M has an initialized accepting run over ρ.

A finite-state (untimed) process with event set A is modeled using a Büchi automaton over $\mathcal{P}^+(A)$ (it is convenient to identify the empty event set with ε). S corresponds to the states of the system, and E gives its transitions. The purpose of the accepting set F is to restrict attention to only the *fair* runs. For example, in a system with two processes we consider only those computation sequences in which each process executes infinitely often.

The class $\mathcal{C}$ of *finite-state processes* comprises of processes of the form (A, X) where $X = L(\text{M})$ for some Büchi automaton M over $\mathcal{P}^+(A)$.

Given a Büchi automaton M we can construct a finite automaton which accepts precisely the finite words accepted by M, and an ε-free Büchi automaton accepting precisely the infinite words accepted by M. From the known results it follows that Büchi automata are (effectively) closed under union, intersection, projections and complementation ([Ch74, WVS83]). As a consequence we have the following useful results:

Fact: *The class $\mathcal{C}$ of processes is closed under parallel composition, hiding, and renaming.*

Proof: First we consider parallel composition of two processes P_i $(i = 1, 2)$ with event sets A_i and represented as Büchi automata M$_i$. First expand the event set of each automaton to the union of the two event sets. This can be done by replacing each edge $\langle s, s', \sigma \rangle$ of M$_i$ by a set of edges of the form $\langle s, s', \sigma' \rangle$ such that $\sigma' \cap A_i = \sigma$. Now that the alphabet of both the automata is $\mathcal{P}^+(A_1 \cup A_2)$, parallel composition reduces to language intersection, and can be implemented by a product construction for Büchi automata ([WVS83]).

Hiding of an event corresponds to removing it from every edge label in which it appears. Since our automata have ε-edges this poses no problem. Renaming can be done by renaming events in all the edge labels. ■

Fact: *The inclusion problem for the class $\mathcal{C}$ is decidable.*

Proof: To test whether the language of one automaton is contained in the other, we check for emptiness of the intersection of the first automaton with the complement of the second, using known constructions for complementing Büchi automata ([SVW87, Sa88]). To test for emptiness, we need to search for a cycle that is reachable from a start state and includes at least one final state. ■

Complementing a Büchi automaton involves an exponential blow-up in the number of states, and hence the complexity of checking for inclusion is exponential. However, checking whether the language of one automaton is contained in the language of a *deterministic* automaton can be done in polynomial time ([CDK89]).

An ω-automaton M is *deterministic* iff there is a single start state, and for every state s and every symbol σ, the number of edges starting at s and labeled with either σ or ε is at most 1. Thus, a deterministic automaton has at most one run over any word. It turns out that, unlike the automata on finite strings, the class of languages accepted by deterministic Büchi automata is strictly smaller than that accepted by their non-deterministic counterparts. Muller automata defined below avoid this problem at the cost of a more complicated acceptance condition.

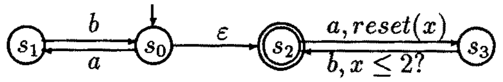
A *Muller automaton* M is an ω-automaton with an acceptance family $\mathcal{F} \subseteq 2^S$. A run r of M over a word ρ is an *accepting run* iff the set of states repeating infinitely often along r equals some set in $\mathcal{F}$. The language accepted by M is defined as in case of Büchi automata.

The class of languages accepted by Muller automata is the same as that accepted by Büchi automata, and also equals that accepted by deterministic Muller automata. Algorithms for constructing the intersection of two Muller automata and for checking the language inclusion are known ([CDK89]). Hence, one can possibly use deterministic Muller automata as a representation for the processes in $\mathcal{C}$.

3.2 Timed automata

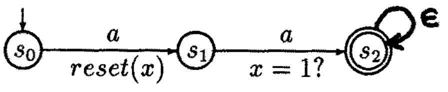
In this section we extend ω-automata to timed automata accepting timed languages. With each ω-automaton we associate a finite set of (real-valued) *clocks*. A clock can be set to zero simultaneously with any transition of the automaton. At any instant, the reading of a clock equals the time elapsed since the last time it was set. With each transition we associate an enabling condition which compares the current values of the clocks with time constants. Before we define the timed automata formally, let us consider some examples of timed Büchi automata.

Example 1 : The following automaton over the alphabet $\{a, b\}$ accepts the timed language $\{((ab)^\omega, \tau) : \exists i. \forall j \geq i.\ \tau_{2j+1} \leq \tau_{2j} + 2\}$.



The start state is s_0, s_2 is the accepting state, and there is a single clock x. The clock is set on the transition from s_2 to s_3, and the transition from s_3 to s_2 is enabled only if the time elapsed since then is not greater than 2. Interpreting b as the response to a request a, the automaton models a system in which the response time is eventually always less than 2 seconds. ■

Example 2 : The language accepted by the following automaton over $\{a\}$ is $\{(a^\omega, \tau) : \exists i, j : \tau_j = \tau_i + 1\}$.



The start state is s_0, and s_2 is the accepting state. ■

Thus the mechanism of resetting a clock with one transition, and associating with another transition an enabling condition which compares the value of this clock with some time constant, expresses a bound on the delay between the two transitions. Note that clocks can be set asynchronously of each other. Also arbitrarily many events can occur in a finite interval of time. The finiteness of the number of clocks corresponds to the assumption that the future behavior of a process depends on only a finite number of delays.

Before we give the definition of timed automata, we need to develop some notation. Let $\mathrm{N} = \{0, 1, 2 \ldots\}$ be the set of time constants. For a set X of clocks, $\Phi(X)$ denotes the set of formulas constructed from the atomic formulas of the form $x \leq c$ or $c \leq x$, where $x \in X$ and $c \in \mathrm{N}$, using the logical connectives. A *time assignment* for X assigns a real value to each clock. The formulas in $\Phi(X)$ can be interpreted over the time assignments for X in the obvious way. For a time assignment ν for X, and $t \in \mathbf{R}$, $\nu + t$ denotes another time assignment which maps a clock x in X to the value $\nu(x) + t$. For $X' \subseteq X$, $[X' \mapsto t]\nu$ denotes an assignment for X which assigns t to each $x \in X'$, and agrees with ν over the rest of the clocks.

A *timed automaton* is a tuple $\langle \Sigma, \mathrm{S}, \mathrm{S}_0, \mathrm{C}, \mathrm{E} \rangle$, where Σ, S, and S_0 are the same as in an ω-automaton, C is a finite set of clocks, and $\mathrm{E} \subseteq \mathrm{S} \times \mathrm{S} \times [\Sigma \cup \{\varepsilon\}] \times 2^{\mathrm{C}} \times \Phi(\mathrm{C})$ gives the set of transitions. An edge $\langle s, s', \sigma, \lambda, \delta \rangle$ represents a transition from state s to state s' on input symbol σ. λ gives the set of clocks to be reset with this transition, and δ gives the enabling condition. An ω-automaton is a special case of a timed automaton with an empty set of clocks.

The automaton starts in one of the start states with all the clocks initialized to 0. As time advances the values of all the clocks change reflecting the elapsed time. At any point in time, the automaton can change state through a transition $\langle s, s', \sigma, \lambda, \delta \rangle$ reading the input σ, provided the current values of the clocks satisfy δ. With this transition the clocks in λ get reset to 0, and thus start counting time with respect to it. We now define a run of a timed automaton.

Given a timed trace (ρ, τ) over Σ, we say that $r : \langle s_0, \nu_0, 0 \rangle \xrightarrow{\rho'_0} \langle s_1, \nu_1, t_1 \rangle \xrightarrow{\rho'_1} \langle s_2, \nu_2, t_2 \rangle \cdots$ is a run of M over (ρ, τ), provided

- the input word read is ρ: ρ' is an infinite word over $[\Sigma \cup \{\varepsilon\}]$ such that ρ is ρ' with all ε's removed,
- r satisfies proper consecution requirements: for each i, there is an edge in E of the form $\langle s_i, s_{i+1}, \rho'_i, \lambda_i, \delta_i \rangle$ such that the time assignment $\nu_i + t_{i+1} - t_i$ satisfies δ_i and ν_{i+1} equals $[\lambda_i \mapsto 0](\nu_i + t_{i+1} - t_i)$, and
- time progresses along r: for every $t \in \mathbf{R}$, for some $i \geq 0$, $t_i > t$.

Furthermore, r is called an *initialized run* if it is properly initialized: s_0 is a start state, and for each $x \in \mathrm{C}$, $\nu_0(x) = 0$.

We can couple acceptance criteria with timed automata, and use them to define timed languages. A *timed Büchi automaton* (TBA for short) is a timed automaton with a set $\mathrm{F} \subseteq \mathrm{S}$ of accepting states. A run r of a TBA over a timed trace is called an *accepting run* iff for infinitely many i's, $s_i \in \mathrm{F}$. For a TBA M, the language $L(\mathrm{M})$ of timed traces it accepts, is defined to be the set $\{(\rho, \tau)$: M has an initialized accepting run over $(\rho, \tau)\}$.

We model finite-state real-time processes using TBAs. We consider the class $\mathcal{D}$ of *timed finite-state processes*, those of the form (A, L), where L is the language accepted by some TBA over $\mathcal{P}^+(A)$.

The next theorem considers some closure properties of TBAs.

Theorem: *The class of timed languages accepted by TBAs is closed under union, intersection, and projections.*

Proof: First consider the boolean operations over two TBAs M_i, $i = 1, 2$. Assume without loss of generality that the sets C_1 and C_2 are disjoint. In each case, the resulting set of clocks is $\mathrm{C}_1 \cup \mathrm{C}_2$. Union corresponds to the disjoint union of the two automata. Intersection can be implemented by modifying the product construction for Büchi automata. For a joint transition of the resulting automaton corresponding to the M_1-transition $\langle s_1, s_1', a, \lambda_1, \delta_1 \rangle$ and the M_2-transition $\langle s_2, s_2', a, \lambda_2, \delta_2 \rangle$, the set of the clocks to be reset is $\lambda_1 \cup \lambda_2$, and the enabling condition is $\delta_1 \wedge \delta_2$.

Projections are modeled by changing the labels of all the edges appropriately. ∎

It follows that $\mathcal{D}$ is closed under operations of parallel composition, hiding, and renaming. As in case of untimed processes, for parallel composition, we first make the event sets of the two automata the same (using the same technique), and then take language intersection.

We can define timed automata with Muller acceptance condition also. A *timed Muller automaton* (TMA for short) is a timed automaton with acceptance family $\mathcal{F} \subseteq 2^{\mathrm{F}}$ specified as in case of Muller automata. The language accepted by a TMA is defined in the obvious way.

A TBA given by $\langle \Sigma, \mathrm{S}, \mathrm{S}_0, \mathrm{C}, \mathrm{E}, \mathrm{F} \rangle$ can be viewed as a TMA $\langle \Sigma, \mathrm{S}, \mathrm{S}_0, \mathrm{C}, \mathrm{E}, \mathcal{F} \rangle$, where the acceptance family is given by $\mathcal{F} = \{\mathrm{S}' \subseteq \mathrm{S} : \mathrm{S}' \cap \mathrm{F} \neq \phi\}$. On the other hand, given a TMA, we can construct a TBA accepting the same language using the simulation of Muller acceptance condition by Büchi automata. It follows that the class of timed languages accepted by TMAs is the same as that accepted by TBAs.

3.3 Emptiness

The information about the timing constraints provided by a timed automaton can be used to rule out certain behaviors, and to prove that some qualitative temporal specification is met. We will show that a Büchi automaton can be constructed that accepts exactly the set of untimed traces that are consistent with the timed traces accepted by a timed automaton.

Let $\mathrm{M} = \langle \Sigma, \mathrm{S}, \mathrm{S}_0, \mathrm{C}, \mathrm{E}, \mathrm{F} \rangle$ be a TBA. A *snapshot* of M is a pair consisting of the state of the automaton and a time assignment giving the current values of all its clocks. A snapshot has enough information to determine which transitions will be enabled. Since the number of such snapshots is infinite, we cannot possibly build an automaton whose states are snapshots of M. But if two snapshots with the same M-state agree on the integral parts of all the clock values, and also on the ordering of the fractional parts of all the clocks, then the future behaviors starting from the two snapshots are very similar. The integral parts of the clock values are needed to determine whether or not a particular enabling condition is met, whereas the ordering of the fractional parts is needed to decide which clock will change its integral part first. For example, if two clocks x and y are between 0 and 1 in a snapshot, then a transition with enabling condition $x = 1$ can be followed by a transition with enabling condition $y = 1$, depending on whether or not the snapshot satisfies $x < y$.

The integral readings of the clocks can get arbitrarily large. But if a clock x is never compared with a constant greater than c, then its actual value, once it exceeds c, is of no consequence in deciding the allowed paths.

Now we formalize this notion. For each $x \in \mathrm{C}$, let c_x be the largest constant that x is compared with in any enabling condition. For any $t \in \mathbf{R}$, $\mathit{fract}(t)$ denotes the fractional part of t, and $\lfloor t \rfloor$ denotes the integral part of t.

Given time assignments $\nu, \nu' \in [\mathrm{C} \to \mathbf{R}]$, let us say $\nu \cong \nu'$ iff the following hold:

- For each $x \in \mathrm{C}$, either $\lfloor \nu(x) \rfloor$ and $\lfloor \nu'(x) \rfloor$ are the same, or both $\nu(x)$ and $\nu'(x)$ are greater than c_x.
- For every $x, y \in \mathrm{C}$ such that $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, $\mathit{fract}(\nu(x)) \leq \mathit{fract}(\nu(y))$ iff $\mathit{fract}(\nu'(x)) \leq \mathit{fract}(\nu'(y))$.

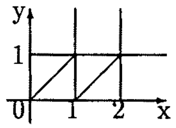
Lemma: *Let s be any state, and ν, ν' be time assignments such that $\nu \cong \nu'$. There is an accepting run starting at $\langle s, \nu, 0 \rangle$ iff there is an accepting run starting at $\langle s, \nu', 0 \rangle$.* ∎

A stronger version of the above lemma is proved in [ACD90].

We will use $[\nu]$ to denote the equivalence class of ν with respect to $\cong$. A *region* is a pair $\langle s, [\nu] \rangle$ where s is a state and ν is a time assignment. Note that there are only a finite number of regions. We will define an edge relation over the regions such that the paths in the resulting ω-automaton mimic the runs of M in a certain way. The edge relation captures two different types of events: (1) transitions in M and (2) moving into a new equivalence class of snapshots because of the passage of time. We define a *successor* function over equivalence classes of time assignments to capture the second type of transitions.

Let α, β be distinct equivalence classes of $[\mathrm{C} \to \mathbf{R}]$. $\mathit{succ}(\alpha) = \beta$ iff for each $\nu \in \alpha$, there exists a positive $t \in \mathbf{R}$ such that $\nu + t \in \beta$, and for all $t' < t$, $\nu + t' \in \alpha \cup \beta$.

Let us consider an example with two clocks x and y with $c_x = 2$ and $c_y = 1$. The equivalence classes are shown in the following figure:



Equivalence classes:

Corner points: e.g. (0,1)
Open line segments: e.g. $\{(x, y) : (0 < x < 1) \wedge (x = y)\}$
Open regions: e.g. $\{(x, y) : 0 < x < y < 1\}$

The successor of any class α is the class to be hit first by a line drawn from some point in α in the diagonally upwards direction. For example, the successor of the class $\{(1, 0)\}$ is the class $\{(x, y) : (1 < x < 2) \wedge (y = x - 1)\}$. The successor of $\{(x, y) : 0 < y < x < 1\}$ is the class $\{(1, y) : 0 < y < 1\}$.

Let us call an equivalence class α a *boundary class*, if for each $\nu \in \alpha$ and for any positive t, ν and $\nu + t$ are not equivalent. In the above example, the classes which lie on either horizontal or vertical lines are boundary classes.

Now we can construct the desired automaton:

Theorem: *Given a TBA* $\mathrm{M} = \langle \Sigma, \mathrm{S}, \mathrm{S}_0, \mathrm{C}, \mathrm{E}, \mathrm{F} \rangle$, *there exists a Büchi automaton* M' *over* Σ *such that* $L(\mathrm{M}') = \mathit{Untime}[L(\mathrm{M})]$.

Proof: First we construct an ω-automaton M''. States of M'' are the regions of M. The start states of M'' are of the form $\langle s_0, [\nu_0] \rangle$ where s_0 is a start state of M, and for each $x \in \mathrm{C}$, $\nu_0(x) = 0$. The edge set consists of two types of edges:

- Edges representing the passage of time: Each state $\langle s, \alpha \rangle$ has an ε-labeled edge to $\langle s, succ(\alpha) \rangle$.
- Edges representing the transitions of M: For each edge $\langle s, s', \sigma, \lambda, \delta \rangle$ of M, there is a σ-labeled edge from $\langle s, \alpha \rangle$ to $\langle s', [[\lambda \mapsto 0]\nu] \rangle$, provided ν satisfies the enabling condition δ, and either $succ(\alpha) = [\nu]$, or $\nu \in \alpha$ and α is not a boundary class.

There is a simple correspondence between the runs of M and runs of M$''$. Let $r : \{\langle s_i, \nu_i, t_i \rangle : i \geq 1\}$ be any run of M. For each $i \geq 1$, we can find a finite path r_i in M$''$ joining $\langle s_i, [\nu_i] \rangle$ to $\langle s_{i+1}, [\nu_{i+1}] \rangle$ by taking successor classes of $[\nu_i]$ as time increases from t_i to t_{i+1}. Thus the run r' obtained by concatenating the segments r_i corresponds to the run r in a natural way. Similarly, given a run of M$''$ we can construct a corresponding run of M.

If r is an accepting run of M, that is, some state s from F repeats infinitely often, then for some equivalence class α, the state $\langle s, \alpha \rangle$ repeats infinitely often along r'. Let F$'' = \{\langle s, \alpha \rangle : s \in \mathrm{F}\}$.

Since time progresses without bound along r, every clock $y \in$ C is either reset infinitely often or eventually it always increases. Hence, for each $y \in$ C, along r', infinitely many regions satisfy either $y = 0$, or $y > c_y$.

Let us denote the set $\{\langle s, [\nu] \rangle : s \in \mathrm{S} \wedge (\nu(y) = 0 \vee \nu(y) > c_y)\}$ by F_y, for $y \in$ C. Let us call a run of M$''$ accepting iff it visits some state from F$''$, and some state from each of F_y's infinitely often. It follows that L(M$''$) equals *Untime*[L(M)].

The acceptance condition of M$''$ is not a Büchi acceptance condition, but it is straightforward to construct a Büchi automaton M$'$ accepting the same trace language. ∎

From the theorem it follows that for all processes $P \in \mathcal{D}$, *Untime*$(P) \in \mathcal{C}$. Furthermore, given a TBA, to check whether its language is empty, we can check for the emptiness of the language of the corresponding Büchi automaton constructed by the theorem. The next corollary follows.

Corollary: *Given a timed Büchi automaton* M *there is a decision procedure to check for the emptiness of* L(M). ∎

The size of M$'$ constructed by the above proof is $O(|\mathrm{C}|! \cdot (|\mathrm{S}| + |\mathrm{E}|) \cdot 2^{|\delta|})$, where $|\delta|$ is the total number of bits used in the binary encoding of the constants in the enabling conditions. Thus the complexity of deciding emptiness of a TBA is exponential in the number of clocks and the length of its timing constraints. This blow-up in complexity seems unavoidable; we can reduce the acceptance problem for linear bounded automata to the emptiness question for TBAs to prove that the problem of deciding whether or not the language of a TBA is empty is PSPACE-complete. Note that the source of this complexity is not the choice of **R** to model time. The same result can be proved if we leave the syntax of timed automata unchanged, but use the discrete integer domain to model time.

The above methods can be used even if we change the acceptance condition for timed automata. In particular, given a TMA M we can effectively construct a Büchi automaton which accepts *Untime*[L(M)], and use it to check for the emptiness of L(M).

3.4 Inclusion

For verification, one may describe the implementation as a TBA, I, and the specification as another TBA, S. Then one way to define verification is: I satisfies S iff $L(I) \subseteq L(S)$. However, there is a stumbling block to automatic verification by this approach:

Theorem: *Given timed Büchi automata* M *and* M′*, the problem of deciding whether* $L(\mathrm{M}) \subseteq L(\mathrm{M}')$ *is undecidable — in fact,* Σ_1^1*-hard.*
Proof: We reduce the question of deciding whether a non-deterministic 2-counter machine has a *recurring* computation (a computation in which its start state appears infinitely often), to the language inclusion problem of timed automata.

Consider a machine with counters α and β, and a program with m labeled instructions that can increment/decrement the counters, jump based on testing for a counter being zero, and can choose nondeterministically between two instructions. A configuration of the machine is represented by the triple $\langle i, c, d\rangle$, where i gives the value of the location counter, and c and d give the values of the counters α and β, respectively. We encode the computations of the machine using timed traces over the alphabet $\{b_1, \ldots b_m, a_1, a_2\}$. A configuration $\langle i, c, d\rangle$ is represented by the event sequence $b_i a_1^c a_2^d$. We say that a timed trace encodes a computation of the machine, if the sequence of events happening in the time interval $[j, j+1)$ encodes the j-th configuration of the computation, for each $j \geq 0$. Note that denseness of the underlying time domain allows the counter values to get arbitrarily large.

We construct a boolean expression S of TBAs which accepts precisely the computations of the machine. Using a TBA we can express the nature of the initial configuration. We need to express that the successive configurations are related as per the requirements of the program instructions. The question reduces to whether timed automata can be used to express that the number of a_1 (or a_2) events in two intervals encoding the successive configurations is the same (or that the number is one less or one greater). This can be done by requiring that every a_1 in the first interval has a matching a_1 at distance 1 and vice versa. This requirement can be specified as the complement of the TBA requiring some a_1 in the first interval with no matching a_1 at distance 1 or vice versa.

The language of the conjunction of S with a TBA expressing the recurrence requirement is non-empty iff the machine has a recurring computation. Since TBAs are closed under union and intersection, the emptiness question of a boolean expression over timed automata can be phrased as a language inclusion question of two automata. The result follows. ■

This result is not unusual for systems for reasoning about dense real-time. It is similar to the undecidability of logics interpreted over timed traces ([AH89, AD90]). Obviously, the language inclusion problem for TMAs is also undecidable. The undecidability of the inclusion problem has another implication:

Corollary: *Timed Büchi automata are not closed under complement.*
Proof: Given TBAs M and M′, $L(\mathrm{M}) \subseteq L(\mathrm{M}')$ iff $L(\mathrm{M}) \cap L(\mathrm{M}'^c)$ is empty, where M'^c accepts the complement of $L(\mathrm{M}')$. Assume that TBAs are closed under complement. Then it follows that: $L(\mathrm{M}) \not\subseteq L(\mathrm{M}')$ iff there is a TBA N such that $L(\mathrm{M}) \cap L(\mathrm{N})$ is non-empty, but $L(\mathrm{M}') \cap L(\mathrm{N})$ is empty. Since one can effectively enumerate TBAs, construct a TBA accepting the intersection of the languages of two TBAs, and check for emptiness, the complement of inclusion problem is recursively enumerable. This contradicts the Σ_1^1-hardness result. ■

By similar reasoning TMAs also are not closed under complement. To get some insight regarding the non-closure under complementation consider the automaton of example 2. The complement needs to make sure that no pair of a's is separated by distance 1. Since there is no bound on the number of a's that can happen in a time period of length 1, this requires an unbounded number of clocks. It can be proved that the complement language cannot be specified by a timed automaton.

4 Deterministic Timed Automata

The results of the previous section imply that we can build a composite timed automaton representing the behavior of a system, using parallel composition, renaming, and hiding. However, we cannot automatically compare it with a specification presented as a TBA. In this section we define deterministic timed automata, and show that deterministic timed Muller automata (DTMA) are closed under the boolean operations. The ability to complement makes it possible to test whether a timed automaton representing an implementation meets a specification expressed by a DTMA.

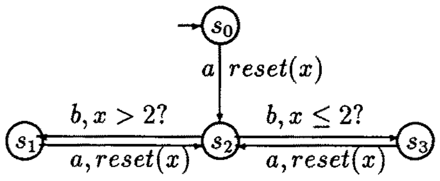
A timed automaton $\langle \Sigma, S, S_0, C, E \rangle$ is called *deterministic*, provided

- it has only one start state, and
- for each state s, for each input symbol σ, for every pair of edges of the form $\langle s, -, \sigma', -, \delta_1 \rangle$ and $\langle s, -, \sigma'', -, \delta_2 \rangle$ such that both the labels σ' and σ'' are in $\{\sigma, \varepsilon\}$, the enabling conditions δ_1 and δ_2 are mutually exclusive (that is, $\delta_1 \wedge \delta_2$ is unsatisfiable).

Thus, a deterministic timed automaton may have two transitions starting at the same state labeled with the same symbol, but given an assignment for the clocks only one of the outgoing transitions can be taken. Consequently, a deterministic timed automaton has at most one initialized run over any given timed trace. This property allows complementing them. Note that in absence of clocks the above definition matches with the definition of determinism for ω-automata.

Since deterministic Büchi automata are strictly less expressive than deterministic Muller automata, deterministic timed Büchi automata (DTBA) are strictly less expressive than deterministic timed Muller automata (DTMA). Furthermore, we will show that DTMAs are closed under complement. Hence, we propose that DTMAs be used as a specification formalism.

Apart from the usual qualitative requirements, and simple timing properties such as bounded response time and periodicity, DTMAs can specify interesting properties involving liveness and timing. One such example property is the convergent response time property of example 1: "The events a and b alternate, and eventually every a is followed by b within time 2." The following DTMA accepts the language $\{((ab)^\omega, \tau) : \exists i, j \geq i : \tau_{2j+2} \leq \tau_{2j+1} + 2\}$.



The start state is s_0. The Muller acceptance family is given by $\{\{s_2, s_3\}\}$. The state s_2 has two mutually exclusive outgoing transitions on b. The acceptance condition requires that the transition with the enabling condition $x > 2$ is taken only finitely often. We believe that no DTBA can specify this property.

A timed automaton $\langle \Sigma, S, S_0, C, E \rangle$ is called *complete* iff for each state s and each input symbol σ, the disjunction of the enabling conditions of the edges starting at s and labeled with σ or ε is a valid formula. Thus some edge is always enabled in a complete automaton. Consequently, a deterministic complete timed automaton has precisely one run over any timed trace.

Given a DTMA we can construct an equivalent complete DTMA easily. First we add a dummy dead state q to the automaton. From each state s, for each symbol σ, we add a σ-labeled edge from s to q with the enabling condition equal to the negation of the disjunction of the enabling conditions of all the edges starting at s and labeled with σ or ε. We leave the acceptance condition unchanged. This construction preserves determinism as well as the set of accepted timed traces.

The next theorem states the closure of DTMAs under the boolean operations.

Theorem: *The class of timed languages accepted by DTMAs is closed under union, intersection, and complementation.*

Proof: Let M_i, for $i = 1, 2$, be two complete DTMAs with disjoint sets of clocks. First we construct a timed automaton M using a product construction. The set of states of M is $S_1 \times S_2$. Its start state is $\langle s_{0_1}, s_{0_2} \rangle$. The set of clocks is $\mathcal{C}_1 \cup \mathcal{C}_2$. For each input symbol σ, corresponding to an M_1-transition $\langle s_1, t_1, \sigma_1, \lambda_1, \delta_1 \rangle$ and an M_2-transition $\langle s_2, t_2, \sigma_2, \lambda_2, \delta_2 \rangle$ such that each label σ_i is either σ or ε, M has a transition $\langle \langle s_1, s_2 \rangle, \langle t_1, t_2 \rangle, \sigma, \lambda_1 \cup \lambda_2, \delta_1 \wedge \delta_2 \rangle$. It is easy to see that M is also deterministic.

Let $\mathcal{F}^1$ consist of the sets $\alpha \subseteq S_1 \times S_2$ such that $\{s\colon \exists s' \in S_2. \langle s, s' \rangle \in \alpha\}$ is in $\mathcal{F}_1$, and $\mathcal{F}^2$ consist of the sets α such that $\{s'\colon \exists s \in S_1. \langle s, s' \rangle \in \alpha\}$ is in $\mathcal{F}_2$. Now coupling M with the Muller acceptance family $\mathcal{F}^1 \cup \mathcal{F}^2$ gives an automaton accepting $L(M_1) \cup L(M_2)$, whereas using the acceptance family $\mathcal{F}^1 \cap \mathcal{F}^2$ gives a DTMA accepting $L(M_1) \cap L(M_2)$.

Finally consider complementation. Let M be a complete DTMA. M has exactly one run over any timed trace. Hence, (ρ, τ) is in the complement of $L(M)$ iff the run of M over it does not meet the acceptance criterion of M. The complement language is, therefore, accepted by a DTMA which has the same underlying timed automaton as M, but its acceptance condition is given by $2^S - \mathcal{F}$. ∎

Since DTMAs are closed under complement, whereas TMAs are not, it follows that the class of languages accepted by DTMAs is strictly smaller than that accepted by TMAs. In particular, the language of example 2, ("some two a's are distance 1 apart") is not representable as a DTMA. Furthermore, DTMAs are not closed under projection operations.

In this paper we have considered only Büchi and Muller acceptance conditions. Other types of acceptance criteria such as Rabin acceptance and Streett acceptance have been studied in the context of ω-automata. Deterministic timed Rabin (or Streett) automata form another complementable subclass of TBAs, possibly different from DTMAs.

Acknowledgements: We thank Tom Henzinger, Harry Lewis, Zohar Manna, and Moshe Vardi for useful discussions.

References

[AK83] S. Aggarwal, R.P. Kurshan, "Modeling elapsed time in protocol specification," *Protocol Specification, Testing and Verification*, III, 1983.

[ACD90] R. Alur, C. Courcoubetis, D.L. Dill, "Model-checking for real-time systems," 5th IEEE LICS, 1990.

[AH90] R. Alur, T.A. Henzinger, "Real-time logics: complexity and expressiveness," 5th IEEE LICS, 1990.

[Bü62] J.R. Büchi, "On a decision method in restricted second-order arithmetic," *Proc. Internat. Congr. Logic, Methodology, and Philosophy of Science 1960*, Stanford Univ. Press, 1962.

[Ch74] Y. Choueka, "Theories of automata on ω-tapes: a simplified approach," *JCSS* **8**, 1974.

[CDK89] E.M. Clarke, I.A. Draghicescu, R.P. Kurshan, "A unified approach for showing language containment and equivalence between various types of ω-automata," Tech. report CMU-CS-89-192, Carnegie Mellon University, 1989.

[CES86] E.M. Clarke, E.A. Emerson, A.P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," ACM *TOPLAS* **8(2)**, 1986.

[Di88] D.L. Dill, *Trace Theory for Automatic Hierarchical Verification of Speed Independent Circuits*, Ph.D. Thesis, Carnegie Mellon Univ., 1988.

[Di89] D.L. Dill, "Timing assumptions and verification of finite-state concurrent systems," *Automatic Verification Methods for Finite State Systems*, *LNCS* **407**, 1989.

[Ho85] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.

[HU79] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata theory, Languages and Computation*, Addison-Wesley, 1979.

[JM86] F. Jahanian, A.K. Mok, "Safety analysis of timing properties in real-time systems," IEEE *Trans. on Software engineering*, **12(9)**, 1986.

[Ko89] R. Koymans, "Specifying message passing and time-critical systems with temporal logic," Ph.D. Thesis, Eindhoven Univ. of Tech., 1989.

[Le89] H.R. Lewis, "Finite-state analysis of asynchronous circuits with bounded temporal uncertainty," Tech. Report TR-15-89, Harvard Univ., 1989.

[Le90] H.R. Lewis, "A logic of concrete time intervals," 5th IEEE LICS, 1990.

[Mc66] R. McNaughton, "Testing and generating infinite sequences by a finite automaton," *Information and Control* **9**, 1966.

[Pn86] A. Pnueli, "Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends," *Current Trends in Concurrency*, *LNCS* **244**, Springer-Verlag, 1986.

[RR86] G.M. Reed, A.W. Roscoe, "A timed model for communicating sequential processes," 13th ICALP, *LNCS* **226**, Springer-Verlag, 1986.

[Sa88] S. Safra, "On the complexity of ω-automata," 29th IEEE FOCS, 1988.

[SVW87] A.P. Sistla, M.Y. Vardi, P. Wolper, "The complementation problem for Büchi automata with applications to temporal logic," *Theoretical Computer Science* **49**, 1987.

[WVS83] P. Wolper, M.Y. Vardi, A.P. Sistla, "Reasoning about infinite computation paths," 24th IEEE FOCS, 1983.