Back to the Future: Towards a Theory of Timed Regular Languages*

Rajeev Alur
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Thomas A. Henzinger†
Computer Science Department
Cornell University
Ithaca, NY 14853

Abstract: *Timed automata* are finite-state machines constrained by timing requirements so that they accept *timed words* — words in which every symbol is labeled with a real-valued time. These automata were designed to lead to a theory of *finite-state real-time properties* with applications to the automatic verification of real-time systems. However, both deterministic and nondeterministic versions suffer from drawbacks: several key problems, such as language inclusion, are undecidable for nondeterministic timed automata, whereas deterministic timed automata lack considerable expressive power when compared to decidable real-time logics.

This is why we introduce *two-way* timed automata — timed automata that can move back and forth while reading a timed word. Two-wayness in its unrestricted form leads, like nondeterminism, to the undecidability of language inclusion. However, if we restrict the number of times an input symbol may be revisited, then two-wayness is both harmless and desirable. We show that the resulting class of *bounded two-way deterministic timed automata* is closed under all boolean operations, has decidable (PSPACE-complete) emptiness and inclusion problems, and subsumes all decidable real-time logics we know.

We obtain a strict hierarchy of real-time properties: deterministic timed automata can accept more languages as the bound on the number of times an input symbol may be revisited is increased. This hierarchy is also enforced by the number of alternations between past and future operators in temporal logic. The combination of our results leads to a decision procedure for a real-time logic with past operators.

1 Introduction

Finite automata are instrumental for the modeling and analysis of many diverse problems within computer science. The universal acceptance of finite automata as the canonical model of finite-state computation can be attributed to the robustness of the model and the appeal of its theory. In particular, the expressions of a variety of competing formalisms — deterministic and nondeterministic finite automata, regular expressions, modal formulas of (extended) temporal logic, and second-order formulas of the monadic theory of one successor (S1S) — are pairwise convertible and their equivalence is decidable. The listed formalisms define the class of *regular languages* (or

*A preliminary version of this paper appeared in the *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science* (FOCS 1992), pp. 177–186.

†In part supported by the National Science Foundation under grant CCR-9200794 and by the United States Air Force Office of Scientific Research under contract F49620-93-1-0056.

ω-regular languages, if the strings in the language are infinite), and we refer to the expressions of these formalism as *finite-state* expressions.

Much effort has been invested in obtaining optimal decision procedures for testing the language inclusion of two finite-state expressions, because these decision procedures lead to algorithms for the verification of concurrent finite-state systems [9, 12, 13, 14, 15]. In a *qualitative* trace model of concurrent computation, the behavior of a system is viewed as a sequence of states or events. Consequently, the possible behaviors of a system represent a formal language, and so do the behaviors that are permitted by the requirement specification of the system. Thus we may use two finite-state expressions A and B to describe a finite-state system and its specification, and we can verify the system by checking that the regular set denoted by A is contained in the regular set denoted by B.[1]

To capture the behavior of *real-time* systems, the qualitative model of computation needs to be augmented by a notion of time. In [2] *timed automata* were proposed as an extension of finite automata for modeling the behavior of concurrent systems over time. Timed automata provide a simple, and yet powerful, way to annotate state-transition graphs with timing constraints using finitely many real-valued variables. A timed automaton accepts *timed words* — strings in which a real-valued time of occurrence is associated with each symbol. The theory of timed automata is useful for the automatic verification of real-time requirements of finite-state systems [1, 2, 3, 11], for synthesizing schedulers from temporal specifications [7], and for solving delay problems [6]. There are, however, problems with timed automata that make us hesitate to accept them as the canonical model for finite-state real-time computation. Unlike in the untimed case, the nondeterministic variety of timed automata is strictly more expressive than the deterministic variety, and both classes have their drawbacks.

Nondeterministic timed automata are too permissive, because their language inclusion problem is undecidable [2]. Thus, for algorithmic verification, one cannot use nondeterministic timed automata for specifying correctness requirements. Also, the class of nondeterministic timed automata is not closed under complementation.

Deterministic timed automata, which have all desirable closure and decidability properties, are too restrictive. Consider, for example, the standard real-time requirement ϕ of timely response, demanding that every environment stimulus p is followed by a system response q after a delay of no less than 5 and no more than 6 time units. Since the time domain is dense, in any interval of unit length an unbounded number of p symbols may occur, and a finite timed automaton cannot "remember" the times of all these p symbols. However, in [3] we showed that it is possible to construct a timed automaton that checks the property ϕ by nondeterministically "guessing" the times of the first and last occurrences of the symbol q in the next 6 unit time intervals. Indeed, as we will show in this paper, the ability of an automaton to "guess" or "predict" the future is necessary for checking timely response: the property ϕ cannot be defined by a deterministic timed automaton.

Yet we cannot put the issue to rest by concluding that even the most basic real-time properties, such as timely response, cannot be verified automatically. On the contrary, the timely-response property ϕ is definable in a simple and decidable real-time logic, MITL [3]:

$$\Box\,(\,p \rightarrow \Diamond_{[5,6]}\; q\,).$$

[1]Usually, the behavior of a system is modeled as an infinite sequence, and the set of all possible behaviors forms an ω-regular language. We are mainly interested in the issues arising due to the introduction of time, and hence, we will consider only finite sequences in this paper. Most of our results can be generalized to infinite sequences in a straightforward manner.

Thus we must conclude, instead, that the step from deterministic to nondeterministic timed automata, with the ensuing undecidability results, is unnecessarily big. Consequently, in [5] we raised the question of finding an intermediate class of automata that

(1) is effectively closed under all boolean operations;
(2) has a decidable language inclusion problem;
(3) is "maximal," at least in the sense that it subsumes both deterministic timed automata and the real-time logic MITL.

This paper presents such a desired class of automata. We define *two-way timed automata* by allowing timed automata to move backward as well as forward "in time" (i.e., over the timed input word). While in the untimed case, the ability to move back and forth over the input word does not change the expressive power of finite automata, deterministic two-way timed automata recognize more languages than their one-way counterparts. Consider, again, the timely-response requirement ϕ that cannot be checked in a deterministic forward manner. Since all the nondeterminism required to check ϕ are some predictions about the future, timely response can be checked in a deterministic way by an automaton that can travel into the future and back. In fact, the property ϕ is particularly easy to check "backwards": go to end of the input word and while coming back, remember the times of the first and the last occurrences of the symbol q for the previous 6 unit intervals and check that the time of every occurrence of p concurs with the requirement ϕ.

Our results will be twofold. First, we study the impact of two-wayness on deterministic and nondeterministic timed automata. We show that the unrestricted ability of an automaton to move both ways makes timed automata too powerful, rendering even the emptiness problem for deterministic automata undecidable. This prompts us to look at the number of trips into the future that an automaton makes when accepting an input word. For every nonnegative integer k, a k*-bounded* timed automaton is defined to be a two-way automaton that visits each symbol of an input word at most $2k+1$ times. We show that this bounded form of two-wayness does not add to the expressiveness of nondeterministic forward automata. In the deterministic case, however, we obtain an infinite strict hierarchy of language classes recognizable by deterministic bounded timed automata, which lies between the language classes accepted by deterministic and nondeterministic forward automata. It is the limit $\cup_k \mathbf{DTA}_k$ of this hierarchy that satisfies the desired closure and decidability properties listed above. We give PSPACE decision procedures for the emptiness and inclusion problems of $\cup_k \mathbf{DTA}_k$ (both problems are PSPACE-hard already for deterministic forward automata).

Second, we study the relationship between the proposed hierarchy of timed automata and real-time logics, and we obtain new decision procedures and verification algorithms. We show that the models of any MITL-formula can be characterized by a deterministic 1-bounded automaton. This translation is much simpler than the translation from MITL to nondeterministic timed automata [3], and it leads to an alternative decision procedure for MITL. Then we introduce a new real-time logic MITL^P, an extension of MITL with *past* temporal operators. We show that any property specified by an MITL^P-formula can also be defined by a deterministic bounded timed automaton, and this translation is used to obtain an EXPSPACE decision procedure for the satisfiability problem of MITL^P (already MITL itself is EXPSPACE-hard). Thus we have a model-checking algorithm for verifying that a system described by a timed automaton (or by an alternative formalism such as timed transition systems [8]) satisfies the correctness requirements specified in MITL^P. Interestingly, there is a close relationship between the number of alternations of past and future temporal operators in an MITL^P-formula ψ and the bound on the two-wayness of any deterministic automaton that checks the property ψ. We show that every additional alternation between past and future operators yields a strictly more expressive real-time specification language — a phenomenon not observed in any other temporal logic.

2 Two-way Timed Automata

Timed words and timed languages

We study formal languages of timed words. Formally, a *timed word* w over an alphabet Σ is a finite sequence $(\sigma_1, \tau_1)(\sigma_2, \tau_2)\ldots(\sigma_n, \tau_n)$ of symbols $\sigma_i \in \Sigma$ paired with real numbers $\tau_i \in \mathrm{R}$ such that the sequence $\overline{\tau} = \tau_1\tau_2\ldots\tau_n$ is strictly monotonically increasing. Sometimes we will denote the timed word w by the pair $(\overline{\sigma}, \overline{\tau})$.

A *timed language* over an alphabet Σ is a set of timed words over Σ. Notice that a timed language can be viewed as a formal language over the uncountable alphabet $\Sigma \times \mathrm{R}$. However, unlike in conventional formal-language theory, the structure of the alphabet is of importance to us: we use operations such as $\leq$ and $+1$ over the reals R. All results of this paper continue to hold if we replace the time domain R by the set of rational numbers Q.

Timed automata

Timed automata were proposed in [2] as an abstract model for finite-state real-time systems. These automata are finite-state machines constrained by timing requirements so that they accept (or generate) timed words. Consequently, every timed automaton defines a timed language. Timed automata, as defined in [2], can read timed words only in the forward direction. Here we study more general timed automata, which can move in both directions and, consequently, visit a symbol of a timed word any number of times.

A timed automaton operates with finite control — a finite set of locations and a finite set of real-valued variables called *clocks*. The automaton may assign to a clock a value chosen randomly from R. The transitions of the automaton put certain constraints on the clock values: a transition may be taken only if the current values of the clocks satisfy the associated constraints. We use a special variable T to denote the time of the current symbol. A *clock constraint* is an arithmetic expression of a simple form: it is a boolean combination of atomic formulas of the form $T \leq x + c$, where x is a clock variable and c is an integer constant.

It is convenient to allow our automata to recognize the end of the input. For a timed word w over the alphabet Σ, let $w\$$ be the extension of w with the pair $(\$, 0)$, where $\$ \notin \Sigma$ is the end-marker (the choice of the time value 0 associated with the end-marker is arbitrary and has no role in any of the results). A timed word w will be presented as $w\$$ to the automaton, and thus the automaton can check for \$ to detect the right end of the input.

A *two-way timed automaton* (2NTA) $\mathcal{A}$ is a tuple $(\Sigma, \mathrm{S}, \mathrm{S}_0, \mathrm{S}_\mathrm{F}, \mathrm{C}, \mathrm{E})$, where

- Σ is a finite alphabet,
- S is a finite set of locations,
- $\mathrm{S}_0 \subseteq \mathrm{S}$ is a set of start locations,
- $\mathrm{S}_\mathrm{F} \subseteq \mathrm{S}$ is a set of accepting locations,
- C is a finite set of clock variables, and
- E is a set of transitions.

Each transition is a tuple $(s, s', \sigma, \mathit{reset}, \mathit{cond}, \mathit{dir})$, with a source location $s \in \mathrm{S}$, a target location $s' \in \mathrm{S}$, an input symbol $\sigma \in \Sigma \cup \{\$\}$, a set $\mathit{reset} \subseteq \mathrm{C}$ of clocks to be reset, a clock constraint cond, and a direction $\mathit{dir} \in \{-, +\}$.

Suppose that the timed automaton $\mathcal{A}$ reads the input $w\$$ for the timed word $w = (\overline{\sigma}, \overline{\tau})$ of length n. While the automaton moves back and forth over the input, its configuration is always determined by the location in which the control resides, the values of all clock variables, and the position of the tape head. Consequently, a *state* of the timed automaton $\mathcal{A}$ is a triple (s, γ, ℓ) consisting of a location $s \in \mathrm{S}$, a map γ from C to R that provides an interpretation for the clocks of the automaton, and a tape position $1 \leq \ell \leq n+1$ that indicates the input symbol being scanned.

The automaton starts in one of its start locations scanning the first symbol (σ_1, τ_1) of the input word $w\$$. The transition relation over the states of the automaton is defined as follows. The automaton can change its state from (s, γ, ℓ) to (s', γ', ℓ') using a transition of the form $e = (s, s', \sigma_\ell, \mathit{reset}, \mathit{cond}, \mathit{dir})$, written $(s, \gamma, \ell) \xrightarrow{e} (s', \gamma', \ell')$, iff

1. If $\mathit{dir} = +$, then $\ell \neq n+1$ and the new position ℓ' of the tape head is $\ell + 1$; otherwise, $\mathit{dir} = -$ and $\ell \neq 1$ and the new position of the head is $\ell - 1$.

2. For each clock x, if $x \notin \mathit{reset}$, the new value $\gamma'(x)$ is the same as the old value $\gamma(x)$; otherwise, the new value $\gamma'(x)$ is an arbitrary real number.

3. The clock constraint *cond* evaluates to true for the interpretation that assigns the new value $\gamma'(x)$ to each clock variable $x \in \mathrm{C}$ and the value τ_ℓ to the time variable T.

If no possible next state exists then the automaton terminates; it accepts or rejects the timed word w depending on whether or not its location upon termination belongs to the set $\mathrm{S_F}$.

The behavior of two-way timed automata is captured formally by defining runs that record the state of the automaton at all transition points. A *run* r of the 2NTA $\mathcal{A} = (\Sigma, \mathrm{S}, \mathrm{S_0}, \mathrm{S_F}, \mathrm{C}, \mathrm{E})$ over the input $w\$$, for a timed word $w = (\overline{\sigma}, \overline{\tau})$ of length n, is a finite sequence

$$(s_0, \gamma_0, \ell_0) \xrightarrow{e_0} (s_1, \gamma_1, \ell_1) \xrightarrow{e_1} \cdots (s_k, \gamma_k, \ell_k) \tag{$\dagger$}$$

of locations $s_i \in \mathrm{S}$, clock interpretations $\gamma_i \colon \mathrm{C} \rightarrow \mathrm{R}$, tape positions $1 \leq \ell_i \leq n+1$, and transitions $e_i \in \mathrm{E}$ that satisfy the following three conditions.

- *Initialization*: $s_0 \in \mathrm{S_0}$ and $\ell_0 = 1$.

- *Consecution*: For all $0 \leq i < k$, the transition relation $(s_i, \gamma_i, \ell_i) \xrightarrow{e_i} (s_{i+1}, \gamma_{i+1}, \ell_{i+1})$ holds.

- *Termination*: For every edge $e \in \mathrm{E}$ and every state (s, γ, ℓ) of the automaton, the transition relation $(s_k, \gamma_k, \ell_k) \xrightarrow{e} (s, \gamma, \ell)$ does not hold.

The run r is *accepting* iff

- *Acceptance*: $s_k \in \mathrm{S_F}$.

The timed language $L(\mathcal{A})$ that is accepted by the automaton $\mathcal{A}$ consists of all timed words w such that $\mathcal{A}$ has an accepting run over the input $w\$$.

Examples

Let us consider some examples of two-way timed automata. In the figures, every transition is annotated with the associated alphabet symbol σ, clock constraint *cond*, and reset clocks *reset*. We omit σ if Σ is a singleton, omit *cond* if it equals *true*, and omit *reset* if it is empty. An annotation such as $x := T$ indicates that $x \in \mathit{reset}$ and $x = T$ is a conjunct of *cond*. The solid lines correspond to transitions with $\mathit{dir} = +$, and dotted lines represent transitions with $\mathit{dir} = -$.

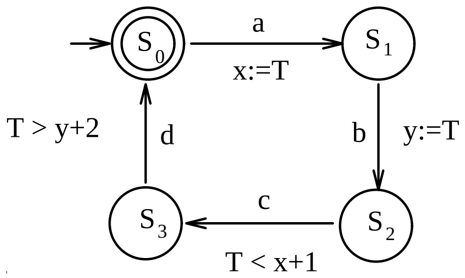

Figure 1: Timed automaton $\mathcal{A}_1$

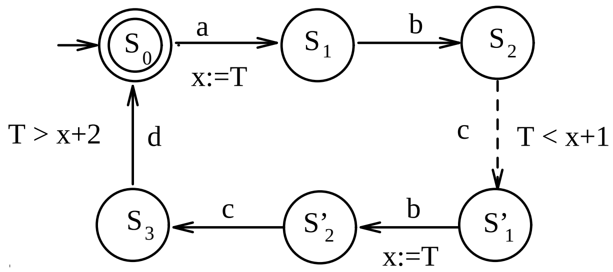

Figure 2: Timed automaton $\mathcal{A}_2$

The timed automaton $\mathcal{A}_1$ of Figure 1 uses two clocks x and y. The location s_0 is the start location, and also the only accepting location. The automaton accepts the timed language $L(\mathcal{A}_1)$ consisting of words $w = ((abcd)^*, \overline{\tau})$ such that for all $j \geq 0$,

$$\tau_{4j+3} < \tau_{4j+1} + 1 \quad \wedge \quad \tau_{4j+4} > \tau_{4j+2} + 2.$$

The automaton loops between the locations s_0, s_1, s_2, and s_3. Whenever $\mathcal{A}_1$ moves from s_0 to s_1 reading a, the clock x gets set to the associated time value. The clock constraint $T < x+1$ associated with the c transition from s_2 to s_3 ensures that c is within 1 time unit of the preceding a. A similar mechanism of setting an independent clock y when reading b and checking its value when reading d, ensures that the time difference between b and the following d is always greater than 2.

The automaton $\mathcal{A}_1$ of Figure 1 always moves forward. The same language $L(\mathcal{A}_1)$ is also accepted by the automaton $\mathcal{A}_2$ of Figure 2 which reverses direction. Similar to $\mathcal{A}_1$, the automaton $\mathcal{A}_2$ sets the clock x while moving from location s_0 to s_1 reading a. But now, it skips over the following b, and checks that the following c is within distance 1 of a. Then the automaton $\mathcal{A}_2$ moves left, back to the b symbol, and sets the clock x to the time associated with b. Subsequently it checks that the following d is at least 2 time units away from b. Thus the ability to move back and forth allows $\mathcal{A}_2$ to enforce the desired constraints using just one clock.

As another example consider the automaton $\mathcal{A}_3$ of Figure 3. The alphabet is unary and the automaton accepts a timed word iff it has two symbols whose times differ by exactly 1.

Deterministic timed automata

In the theory of finite (untimed) automata, deterministic automata form an important subclass. In the untimed case, the property of determinism requires that the input symbols labeling two different transitions with the same source location should be different. This notion was generalized

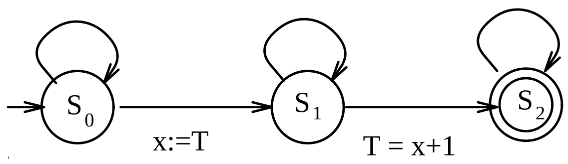

Figure 3: Timed automaton $\mathcal{A}_3$

for (forward) timed automata in [2]. Below we define a similar notion of determinism for two-way timed automata.

A two-way timed automaton $(\Sigma, \mathrm{S}, \mathrm{S}_0, \mathrm{C}, \mathrm{E}, \mathrm{S}_\mathrm{F})$ is *deterministic*, denoted by 2DTA, iff the following conditions hold:

1. There is only one start location; that is, $|\mathrm{S}_0| = 1$.

2. For every transition $(s, s', \sigma, \mathit{reset}, \mathit{cond}, \mathit{dir}) \in \mathrm{E}$ and every reset clock $x \in \mathit{reset}$, the constraint $x = T$ is a conjunct of cond.

3. For all clock interpretations γ, all time values $t \in \mathrm{R}$, and every pair $(s, s_1, \sigma, \mathit{reset}_1, \mathit{cond}_1, \mathit{dir}_1)$ and $(s, s_2, \sigma, \mathit{reset}_2, \mathit{cond}_2, \mathit{dir}_2)$ of transitions, if the clock interpretation that assigns $\gamma(x)$ to all clocks $x \notin \mathit{reset}_1$ and t to T and all clocks $x \in \mathit{reset}_1$, satisfies the constraint cond_1, then the clock interpretation that assigns $\gamma(x)$ to all clocks $x \notin \mathit{reset}_2$ and t to T and all clocks $x \in \mathit{reset}_2$, does not satisfy cond_2.

The first condition ensures that there is a unique way to start the automaton. The second requirement ensures that all resets are deterministic: whenever a clock is reset, there is only one possible choice for its new value, namely, the time of the symbol currently being read. The last condition ensures that the choice of the next transition is uniquely determined by the current state of the automaton. It is easy to check that a deterministic automaton has at most one run over any timed word.

In our examples, the timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic, whereas the automaton $\mathcal{A}_3$ is nondeterministic. In fact, we will show that there is no deterministic two-way timed automaton that accepts the language $L(\mathcal{A}_3)$.

Bounded timed automata

We now proceed to define another restricted class of two-way timed automata. The number of reversals of the run r (see †), denoted by rev_r, is the smallest nonnegative integer m such that for every tape position $1 \leq i \leq n$, the run visits the i-th input symbol at most $2m+1$ times; that is, $\ell_j = i$ for at most $2m+1$ distinct values of $0 \leq j \leq k$.

A two-way timed automaton $\mathcal{A}$ is *k-bounded* iff for every timed word w and every run r of $\mathcal{A}$ over $w\$$, $\mathit{rev}_r \leq k$. We call a k-bounded timed automaton to be of type NTA_k. Furthermore, if it is deterministic, it is of type DTA_k.

Notice that whereas a two-way automaton may loop indefinitely while reading a timed word, the length of a run of a k-bounded automaton over an input word of length n is at most $(2k+1)(n+1)$. It is not difficult to show that the forward timed automata of [2] are of type NTA_0. In our examples, the automaton $\mathcal{A}_1$ is of type DTA_0, and $\mathcal{A}_3$ is of type NTA_0. The automaton $\mathcal{A}_2$ visits every b and c symbol twice; it is of type DTA_1.

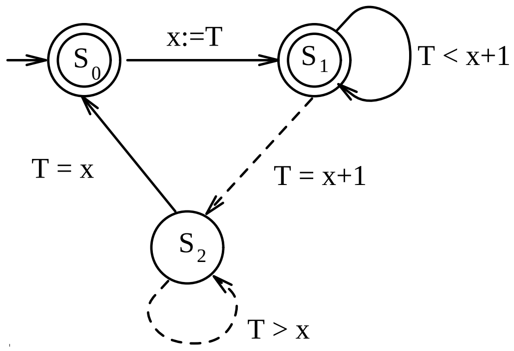

Figure 4: Timed automaton $\mathcal{A}_4$

Not all two-way timed automata need be bounded. As an example, consider the automaton $\mathcal{A}_4$ of Figure 4. Given a timed word $(a^n, \overline{\tau})$ of length $n \geq 0$, the automaton checks at every position $1 \leq i \leq n$ that either τ_j equals $\tau_i + 1$ for some $j \leq n$, or τ_n is smaller than $\tau_i + 1$. The number of visits of $\mathcal{A}_4$ to a symbol is unbounded: to accept a word $(a^n, \overline{\tau}) \in L(\mathcal{A}_4)$ the automaton needs $k-1$ reversals, where k is the maximum number of symbols in a time interval of length 1. In fact, we will show that the language $L(\mathcal{A}_4)$ cannot be accepted by an NTA_k for any $k \geq 0$.

3 The Theory of Bounded Two-way Timed Automata

Any theory of a class of automata must (1) characterize the expressive power of the class, (2) determine under which boolean operations the class is closed, and (3) determine the complexity of the membership, emptiness, universality, and inclusion problems. In this section, we present our results on two-way timed automata with a bounded number of reversals, grouped according to these three categories. We use boldface notation for the set **A** of timed languages that can be defined by automata of type A. Our definition of forward timed automata is slightly more general than the timed automata of [2] (the reset action of [2] is always deterministic); however, all results and proofs of [2] carry over to the classes $\mathbf{DTA}_0$ and $\mathbf{NTA}_0$ in a straightforward way.

Expressiveness

In the untimed case it is well-known that all variants of finite-state machines define the class of regular languages: neither nondeterminism nor the ability to move back and forth add expressive power to finite-state machines [9]. In the timed case, nondeterminism was shown to increase the expressive power of timed automata [2]: $\mathbf{DTA}_0 \subset \mathbf{NTA}_0$. Now we show that a bounded number of reversals increases the expressive power of deterministic timed automata. In fact, each deterministic pass over the input allows the recognition of more timed languages.

Theorem 3.1 $\mathbf{DTA}_k \subset \mathbf{DTA}_{k+1}$.

Proof. It suffices to consider only timed words over the unary alphabet. For a timed word w, define a sequence of values as follows. Let $v_0(w)$ be 1. For $k > 0$, let $u_k(w)$ be the time of the last symbol in the interval $(1, 1 + v_{k-1}(w))$, and let $v_k(w)$ be the time of the last symbol in the interval $(0, u_k(w) - 1)$. Now we define a family of languages L_k, for $k > 0$, such that $w \in L_k$ iff w has a symbol in the interval $(1, 1 + v_k(w))$. We claim that, for all $k > 0$, the language L_k belongs to $\mathbf{DTA}_k$ but not to $\mathbf{DTA}_{k-1}$.

The proof that $L_k \notin \mathbf{DTA}_{k-1}$ given in the full paper; here we outline the argument for $k = 1$. Assume, to the contrary, that the forward DTA_0 $\mathcal{A}$ with m locations defines the language L_1. Suppose that a timed word w contains more than m symbols in both the intervals $(0, v_1(w))$ and $(u_1(w) - 1, 1)$. Using the fact that $\mathcal{A}$ is deterministic and has only m locations, it can be shown that after $\mathcal{A}$ reads the prefix of w up to time 1, no clock is set to the time $v_1(w)$. It follows that $\mathcal{A}$ cannot determine correctly if the interval $(1, 1 + v_1(w))$ contains a symbol. For we can construct another word w' such that w' agrees with w up to time 1 but w' is in L_1 iff w is not. Since $\mathcal{A}$ either accepts or rejects both words, we arrive at a contradiction. ■

On the other hand, any bounded number of reversals can be simulated in a single nondeterministic forward pass.

Theorem 3.2 $\cup_k \mathbf{NTA}_k = \mathbf{NTA}_0$.

Proof. Let $\mathcal{A}$ be an NTA_k. We construct a forward NTA_0 $\mathcal{A}_0$ that accepts the same language. The formal construction is given in the full paper; here we outline the construction under the simplifying assumption that the automaton $\mathcal{A}$ makes one complete forward pass over the input word followed by one complete backward pass, and then stops. The automaton $\mathcal{A}_0$ must simulate both passes of $\mathcal{A}$ in a single forward pass. For every clock x of $\mathcal{A}$, $\mathcal{A}_0$ uses two copies x_f and x_b. A location of $\mathcal{A}_0$ is of the form $(q_f, e_b, \overline{b})$, where q_f records the location of $\mathcal{A}$ on the forward pass, e_b is the transition used by $\mathcal{A}$ on the backward pass, and $\overline{b}$ contains a 3-valued bit b_x for every clock x of $\mathcal{A}$. The interpretation of these bits is as follows: while reading the i-th symbol of w, the bit b_x is 0 if $\mathcal{A}$ resets the clock x at some j-th symbol, $j > i$, on one of the two passes; otherwise, b_x is 1 if $\mathcal{A}$ resets the clock x at some j-th symbol, $j \leq i$, on the forward pass and for all $j \leq j' \leq i$, it does not reset x at the j'-th symbol on the backward pass; otherwise, b_x is 2. A single transition of $\mathcal{A}_0$ corresponds to a forward and a backward transition of $\mathcal{A}$. If $b_x = 0$ then the clock x_f is used for checking the clock constraint of the forward transition and x_b is used for the constraint of the backward transition; if $b_x = 1$ then the clock x_b is used for both the constraints; otherwise the clock x_f is used for both. ■

This gives us the following hierarchy:

$$\mathbf{DTA}_0 \subset \mathbf{DTA}_1 \cdots \subset \cup_k \mathbf{DTA}_k \subset \mathbf{NTA}_0 = \cup_k \mathbf{NTA}_k.$$

The strictness of the last inclusion follows from the closure properties of the two classes (to be considered shortly): $\cup_k \mathbf{DTA}_k$ is closed under complementation but $\mathbf{NTA}_0$ is not.

Closure properties

Untimed finite-state machines are effectively closed under all boolean operations [9]. From the results of [2] it follows that the class $\mathbf{DTA}_0$ is closed under both intersection and complementation, but the class $\mathbf{NTA}_0$ is closed under intersection only. Now we consider these properties for bounded deterministic automata.

Theorem 3.3 *For all $k \geq 1$, the class $\mathbf{DTA}_k$ is closed under complementation, but not under intersection.*

Proof. For bounded deterministic automata, complementation corresponds essentially to complementing the accepting set. The proof of nonclosure under intersection is similar to the proof of Theorem 3.1. ■

However, the limit class $\cup_k \mathbf{DTA}_k$ has the desired closure properties.

Corollary 3.1 $\cup_k\mathbf{DTA}_k$ *is closed under all boolean operations.*

Proof. The closure under complement follows from Theorem 3.3. Given a DTA_{k_1} $\mathcal{A}_1$ and a DTA_{k_2} $\mathcal{A}_2$, the product 2DTA $\mathcal{A}_1 \times \mathcal{A}_2$ simulates first $\mathcal{A}_1$ and then A_2. The product automaton is $(k_1 + k_2 + 1)$-bounded. ■

Complexity issues

In this paper, we consider four problems of automata theory: membership, emptiness, universality, and inclusion. All four problems are decidable for untimed finite-state machines [9]. We know that the membership and emptiness problems are PSPACE-complete for $\mathbf{NTA}_0$ [2]; from our proof of Theorem 3.2, we can conclude that membership and emptiness are in PSPACE for $\mathbf{NTA}_k$ for all $k \geq 0$. The universality and inclusion problems are PSPACE-complete for $\mathbf{DTA}_0$, but undecidable for $\mathbf{NTA}_0$ [2]. Here we consider these questions for $\mathbf{DTA}_k$. The complementation of a DTA_k, its translation into an equivalent NTA_0, and the decision procedure for checking the emptiness of an NTA_0, can be combined in a straightforward manner to obtain the following result.

Theorem 3.4 *For all $k \geq 0$, the membership problem for $\mathbf{DTA}_k$ is solvable in PTIME and the emptiness, universality, and inclusion problems for $\mathbf{DTA}_k$ are PSPACE-complete.*

4 Real-time Logics

In this section we examine the relationship between timed automata and real-time logics.

Bounded-operator temporal logic

We consider the logic MITL, *Metric Interval Temporal Logic*, of [3]. MITL is an extension of the propositional linear temporal logic PTL for writing real-time requirements. It admits time-bounded temporal operators such as $\Diamond_{[2,4]}$, meaning "eventually within time 2 to 4." Formally, given a set of propositions P, the formulas ϕ of MITL are inductively built using boolean connectives and the time-bounded *until* operator:

$$\phi \ := \ p \,|\, \neg\phi \,|\, \phi_1 \wedge \phi_2 \,|\, \phi_1 \, \mathcal{U}_I \, \phi_2,$$

where $p \in \mathrm{P}$ and I is a *nonsingular* interval of the real-line with integer end-points. The subscripting interval I may be open, half-open, or closed; bounded or unbounded. The restriction of nonsingularity, which disallows operators such as $\mathcal{U}_{[3,3]}$, is a necessary condition on the decidability of the satisfiability problem for MITL.

The formulas of MITL are interpreted over timed words over the alphabet 2^{P}.[2] Given an MITL-formula ϕ, a timed word $w = (\overline{\sigma}, \overline{\tau})$ of length n over the alphabet 2^{P}, and a position $1 \leq i \leq n$, the satisfaction relation $(w, i) \models \phi$ is defined inductively as follows:

$(w, i) \models p$ iff $p \in \sigma_i$;
$(w, i) \models \neg\phi$ iff $(w, i) \not\models \phi$;
$(w, i) \models \phi_1 \wedge \phi_2$ iff $(w, i) \models \phi_1$ and $(w, i) \models \phi_2$;
$(w, i) \models \phi_1 \, \mathcal{U}_I \, \phi_2$ iff $(w, j) \models \phi_2$ for some $i \leq j \leq n$ such that $\tau_j \in \tau_i + I$, and $(w, k) \models \phi_1$ for all $i \leq k < j$.

[2] In [3], MITL-formulas are interpreted over *timed state sequences*: a timed state sequence is an infinite sequence $(\sigma_1, I_1)(\sigma_2, I_2)\ldots$ of states $\sigma_i \subseteq \mathrm{P}$ paired with intervals I_i such that the sequence $I_1 I_2 \ldots$ of intervals forms a partition of the real line. Here, our objective is to demonstrate the connection between MITL and timed automata, and to simplify the presentation we have adopted the semantics of timed words.

The timed word w satisfies the formula ϕ (w is a *model* of ϕ) iff $(w,1) \models \phi$. The timed language $L(\phi)$ defined by ϕ consists of the timed words w that satisfy ϕ. Let **MITL** be the set of timed languages definable by MITL-formulas; that is, a timed language L is in **MITL** iff L equals $L(\phi)$ for some MITL-formula ϕ.

We will use some standard abbreviations for additional temporal operators. The defined operators $\Diamond_I \phi$ (time-bounded *eventually*) and $\Box_I \phi$ (time-bounded *always*) stand for *true* $\mathcal{U}_I \phi$ and $\neg\Diamond_I \neg\phi$, respectively.

From temporal logic to timed automata

In [3] we gave an EXPSPACE decision procedure for checking the satisfiability of a given MITL-formula. In particular, we showed how to construct, from a given MITL-formula ϕ, a nondeterministic forward timed automaton $\mathcal{A}_\phi$ that accepts precisely the models of ϕ; that is, $L(\mathcal{A}_\phi) = L(\phi)$. Now we show that we can construct, in fact, a DTA_1 that accepts the models of a given formula.

Consider, for example, the MITL-formula

$$\Box_{[0,1)} (p \rightarrow \Diamond_{[1,2]} q).$$

Let us try to build a deterministic timed automaton that accepts precisely the models of this formula. One possibility is that whenever the automaton reads a p symbol, it stamps a clock with the time of this state, and later checks that within 1 to 2 time units a q symbol is visited. Since the number of p symbols within the interval $[0,1)$ is potentially unbounded, whereas an automaton has only a fixed number of clocks, this simple strategy does not work. In fact, using a proof similar to that of Theorem 3.1, we can show that the above property does not belong to $\mathbf{DTA}_0$.

The solution becomes apparent from the following observation. Let t_1 be the time of the last q symbol within the interval $[1,2)$, and let t_2 be the time of the first q symbol within the interval $[2,3)$. The formula $\Diamond_{[1,2]} q$ holds during the intervals $[0, t_1 - 1]$ and $[t_2 - 2, 1)$, and it does not hold during the interval $(t_1 - 1, t_2 - 2)$. Consequently, while reading the p symbols in the interval $[0,1)$, if the automaton can "predict" t_1 and t_2 in advance, the desired property can be checked. Since a two-way timed automaton can travel back and forth on the input, it need not guess a "future" value such as t_1 nondeterministically, but it can compute t_1 deterministically. The automaton first moves forward, finds the last q symbol in the interval $[1,2)$, and sets a clock x to the time of that symbol. Similarly, it sets a clock y to the time of the first q symbol in the interval $[2,3)$. Then it comes back and checks that no p symbols lie in the "bad" interval $(x-1, y-2)$. This strategy gives us a DTA_1.

Theorem 4.1 *For every MITL-formula ϕ, there exists a* DTA_1 $\mathcal{A}_\phi$ *over the alphabet* 2^{P} *that accepts precisely the models of ϕ.*

Proof. Let ϕ be a formula of MITL and let k_{max} be the maximum of the right end-points of all the intervals appearing in ϕ. The automaton $\mathcal{A}_\phi$ has $2k_{max}$ clocks for each subformula of ϕ. To determine the truth of ϕ with respect to a timed word w at a symbol in the unit time interval $[t, t+1)$, it suffices to know (1) the truth of all subformulas of ϕ at the successor symbol, and (2) for each subformula ψ of ϕ and each interval $[t+i, t+i+1)$ with $0 \leq i \leq k_{max}$, the first and last times that ψ holds in $[t+i, t+i+1)$. The automaton $\mathcal{A}_\phi$ first moves forward to the end of the input and then backtracks. On the return pass, it remembers the finite information (1) in its location and the finite information (2) in its clocks. Thus $\mathcal{A}_\phi$ can determine the truth of every subformula of ϕ at each position on its backwards pass. The formal construction of the automaton is given in the full paper. ■

This gives us $\mathbf{DTA}_0 \not\supseteq \mathbf{MITL} \subset \mathbf{DTA}_1$.

Temporal logic with past operators

Researchers have considered extensions of PTL with past temporal operators such as *since* [10]. These operators can be added to PTL at no extra cost, and though they do not increase its expressive power, they allow a more direct and convenient expression of many properties. We consider a similar extension of MITL.

Let MITL^P be the logic that results from MITL by adding the following clause to the inductive definition of formulas: "If ϕ_1 and ϕ_2 are formulas, then so is $\phi_1 \mathcal{S}_I \phi_2$ for a nonsingular interval I with integer end-points." The meaning of the sime-bounded *since* operator is defined as:

$(w, i) \models \phi_1 \mathcal{S}_I \phi_2$ iff $(w, j) \models \phi_2$ for some $0 \leq j \leq i$ such that $\tau_i \in \tau_j + I$, and $(w, k) \models \phi_1$ for all $j < k \leq i$.

The time-bounded versions of the other past operators such as $\diamondminus$ (*sometime in the past*) and $\boxminus$ (*always in the past*) can be defined using the *since* operator.

Now let us consider the problem of constructing a deterministic automaton that accepts the models of a formula of MITL^P. The number of reversals necessary to check the truth of a formula ϕ of MITL^P is closely related to the number of alternations between past and future operators in ϕ. For every subformula ψ of ϕ, we define $level(\psi)$ inductively:

- If ψ is built from propositions using boolean connectives and the *since* operator, then $level(\psi)$ is 1_F.
- If ψ is built from formulas of level k_F using boolean connectives and the *until* operator, then $level(\psi)$ is k_B.
- If ψ is built from formulas of level k_B using boolean connectives and the *since* operator, then $level(\psi)$ is $(k+1)_F$.

For $k \geq 0$, let MITL^P_k be the fragment of MITL^P that consists of all formulas ϕ such that $level(\phi)$ is either k_B or $(k+1)_F$. Note that all formulas of MITL belong to MITL$^P_1$.

Theorem 4.2 $\mathbf{MITL}^P_k \subset \mathbf{DTA}_k$.

Proof. Given a formula ϕ, we construct a deterministic automaton $\mathcal{A}_\phi$ that checks the truth of ϕ as follows. For every level k, on the k-th forward pass the automaton determines the truth of every subformula ψ with $level(\psi) = k_F$. Similarly, on the k-th reverse pass the automaton determines the truth of every subformula ψ with $level(\psi) = k_B$. ■

The satisfiability problem for MITL is known to be EXPSPACE-complete [3]. The translation from MITL^P to bounded DTAs and the decision procedure for solving the emptiness problem for bounded DTAs can be combined to obtain a decision procedure for MITL^P.

Corollary 4.1 *The satisfiability problem for* MITL^P *is EXPSPACE-complete.*

The expressive power of MITL^P increases with the number of switches between the past and future operators. Consider, for instance, the family of languages L_k from the proof of Theorem 3.1. If we consider only those timed words all of whose symbols lie within the interval $[0, 2)$, then the language L_k is defined by the following MITL$^P_{k+1}$-formula ϕ_k:

$$\Diamond_{>1} \, (\diamondminus_{<1} \Diamond_{>1})^k \; true \, .$$

This shows that for all $k > 0$, $\mathbf{MITL}^P_{k+1} \not\subseteq \mathbf{DTA}_{k-1}$. It follows that the switches between the past and future operators lead to a strict hierarchy of MITL^P-formulas.

Corollary 4.2 $\mathbf{MITL}^P_k \subset \mathbf{MITL}^P_{k+2}$.

We conjecture that in general, an automaton cannot check a formula of MITL^P_k with less than k reversals. If this is the case, then the previous corollary can be strengthened to $\mathbf{MITL}^P_k \subset \mathbf{MITL}^P_{k+1}$.

5 Some Results on Unbounded Two-way Timed Automata

In Section 3 we addressed expressiveness, closure, and decidability questions for two-way timed automata with a bounded number of reversals. The same questions can be asked for the unrestricted classes **2DTA** and **2NTA**. These classes are perhaps not useful for algorithmic verification, because we will see that unbounded two-wayness is so powerful as to lead to undecidability even for the emptiness problem of **2DTA**.

To get an insight into the power of 2DTAs let us consider temporal logics with punctuality constraints. Recall that the logic MITL disallows punctuality constraints such as "every p symbol is followed by a q symbol after exactly 1 time unit,"

$$\Box(p \rightarrow \Diamond_{=1}\, q). \qquad (\phi_{=})$$

We showed in [4] that allowing such formulas leads to the undecidability of the satisfiability problem. Let $\text{MITL}_{=}$ be the extension of MITL that allows singular intervals as time bounds on the temporal operators. We show that the models of any formula of $\text{MITL}_{=}$ can be defined by a 2DTA. For instance, it is easy to modify the automaton $\mathcal{A}_4$ of Figure 4 so that it accepts precisely the models of the formula $\phi_{=}$. On the other hand, the models of $\phi_{=}$ cannot be recognized using a bounded number of reversals, because this property does not belong to $\mathbf{NTA}_0$ [2].

Theorem 5.1 *For every* $\text{MITL}_{=}$*-formula* ϕ*, there exists a* 2DTA *over the alphabet* 2^{P} *that accepts precisely the models of* ϕ*.*

Using the undecidability of $\text{MITL}_{=}$ we obtain two corollaries.

Corollary 5.1 *The emptiness problem for* **2DTA** *is undecidable.*

Corollary 5.2 $\cup_k \mathbf{DTA}_k \subset \mathbf{2DTA} \not\subseteq \mathbf{NTA}_0$.

We conjecture that the ability to go back and forth unbounded number of times in a deterministic fashion is not powerful enough to model a nondeterministic pass:

$$\mathbf{NTA}_0 \not\subseteq \mathbf{2DTA}.$$

In particular, we believe that the following $\mathbf{NTA}_0$-language L cannot be accepted by any 2DTA. Let the timed word $(\overline{\sigma}, \overline{\tau})$ of length n over a unary alphabet be in L iff there is a sequence $1 = i_1 \leq i_2 \leq \cdots \leq i_m = n$ of positions such that for all $1 \leq k < m$, there is some $1 \leq j \leq n$ with $\tau_j = \tau_{i_k} + 1$ and either $i_{k+1} = j - 1$ or $i_{k+1} = j + 1$. If there is no 2DTA that accepts L, then nondeterminism and unrestricted two-wayness offer two proper and distinct extensions to the class $\cup_k \mathbf{DTA}_k$, which we isolated as the finite-state-verifiable real-time properties. The relationship between various classes is depicted in Figure 5.

Finally, we show that only linear space is required to simulate the behavior of a two-way timed automaton on a given input.

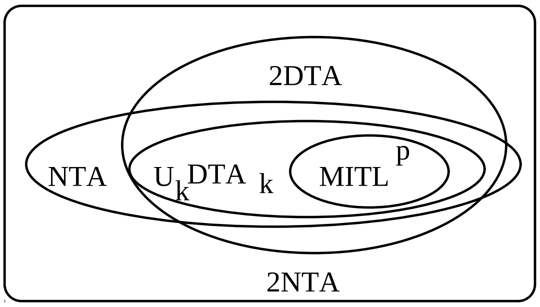

Figure 5: Relationship between various classes

Theorem 5.2 *The membership problem for* **2NTA** *is solvable in PSPACE.*

Proof. Consider a 2NTA $\mathcal{A}$, and let k be the magnitude of the largest constant appearing in a clock constraint in the transitions of $\mathcal{A}$. Given a timed word $w = (\overline{\sigma}, \overline{\tau})$ of length n, define R_w to be the set comprising of $\tau_i + c$ and $(\tau_i + \tau_j + c)/2$ for $1 \leq i, j \leq n$ and $-(k+1) \leq c \leq (k+1)$. Let $\mathcal{A}^*$ be the 2NTA that behaves like $\mathcal{A}$, except that $\mathcal{A}^*$ is required to choose a value from the set R_w whenever a clock is nondeterministically reset. Then $\mathcal{A}$ accepts the timed word w iff $\mathcal{A}^*$ accepts w.

Now we can construct a nondeterministic linear-bounded Turing machine $\mathcal{A}'$ that, given a description of $\mathcal{A}$ and a timed word w, simulates the behavior of $\mathcal{A}^*$ on $w\$$. ■

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. of 5th IEEE Symp. on Logic in Computer Science*, 1990.

[2] R. Alur and D. Dill. Automata for modeling real-time systems. In *Automata, Languages, and Programming: Proc. of 17th ICALP*, LNCS 443. Springer-Verlag, 1990.

[3] R. Alur, T. Feder, and T. Henzinger. The benefits of relaxing punctuality. In *Proc. of 10th ACM Symp. on Principles of Distributed Computing*, 1991.

[4] R. Alur and T. Henzinger. A really temporal logic. In *Proc. of 30th IEEE Symp. on Foundations of Computer Science*, 1989.

[5] R. Alur and T. Henzinger. Time for logic. *SIGACT News*, 22(3), 1991.

[6] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In *Proc. of 3rd Workshop on Computer-Aided Verification*, 1991.

[7] D. Dill and H. Wong-Toi. Synthesizing processes and schedulers from temporal specifications. In *Proc. of 2nd Workshop on Computer-Aided Verification*, 1990.

[8] T. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proc. of 18th ACM Symp. on Principles of Programming Languages*, 1991.

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

[10] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, LNCS 193. Springer-Verlag, 1985.

[11] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. In *Proc. of REX workshop "Real-time: Theory in Practice"*, LNCS 600. Springer-Verlag, 1991.

[12] S. Safra. On the complexity of ω-automata. In *Proc. of 29th IEEE Symp. on Foundations of Computer Science*, 1988.

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

[14] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B. Elsevier Science Publishers, 1990.

[15] P. Wolper, M. Vardi, and A. P. Sistla. Reasoning about infinite computation paths. In *Proc. of the 24th IEEE Symp. on Foundations of Computer Science*, 1983.