Logics and Models of Real Time: A Survey[1]

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

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

Abstract. We survey logic-based and automata-based languages and techniques for the specification and verification of real-time systems. In particular, we discuss three syntactic extensions of temporal logic: time-bounded operators, freeze quantification, and time variables. We also discuss the extension of finite-state machines with clocks and the extension of transition systems with time bounds on the transitions. All of the resulting notations can be interpreted over a variety of different models of time and computation, including linear and branching time, interleaving and true concurrency, discrete and continuous time. For each choice of syntax and semantics, we summarize the results that are known about expressive power, algorithmic finite-state verification, and deductive verification.

1 Introduction

The number of formalisms that purportedly facilitate the modeling, specifying, and proving of timing properties for reactive systems has exploded over the past few years. The authors, who confess to have added to the confusion by advancing a variety of different syntactic and semantic proposals, feel that it would be beneficial to pause for a second — to pause and look back to sort out what has been accomplished and what needs to be done. This paper attempts such a meditation by surveying logic-based and automata-based real-time formalisms and putting them into perspective.

As many of the formalisms that have been promoted in the literature not only suggest different notations, but make radically different assumptions when modeling time and computation, the task of comparing them is often a nontrivial one. We attempt such a comparison by putting, first, all languages on a common semantical ground. Second, we offer a number of common semantical abstractions, and for each abstract semantics, we survey what is known about each language. This two-dimensional analysis of real-time formalisms as syntax-semantics pairs allows us to discuss in the same context different models of computation, such as linear and branching models, as well as different models of time, such as discrete and continuous models. The two-dimensional analysis

[1]An abbreviated version of this paper appeared in *Real Time: Theory in Practice* (J.W. de Bakker, K. Huizing, W.-P. de Roever, G. Rozenberg, eds.), *Lecture Notes in Computer Science* **600**, Springer-Verlag, 1992, pp. 74–106.

also reveals that many meaningful coordinate points have not been addressed in the literature. For some of these, known results can be easily extrapolated (and we do so whenever possible); for others, we pose questions of expressiveness, complexity, and axiomatizability as open problems.

The remainder of the paper is organized in five sections. Section 2 attempts to give a unified semantical framework for real-time systems. The subsequent two sections present syntax. Real-time extensions of temporal logic are summarized in Section 3; real-time extensions of state-transition formalisms, in Section 4. The final two sections survey technical results about these languages. First, in Section 5, we collect all results regarding the impossibility, possibility, and complexity of verifying timing requirements of reactive systems. Second, in Section 6, we review what is known about the absolute and relative expressive power of real-time specification methods.

2 Real-time Semantics

We attempt to define a general semantics for real-time systems. Many previous suggestions are special cases of our definition.

2.1 Concrete semantics

We define a real-time system to be a set of timed state sequences, each of which represents a possible behavior of the system. As time domain, we choose the set of nonnegative real numbers, denoted by R. In a timed state sequence, we use intervals of the real line to specify the duration of system states.

Interval sequences

An interval is a convex subset of R. Intervals may be open, halfopen, or closed; bounded or unbounded. More precisely, every interval is of the form $[a,b]$, $[a,b)$, $[a,\infty)$, $(a,b]$, (a,b), or (a,∞), where $a \leq b$ and $a,b \in \mathrm{R}$ for the left end-point a and the right end-point b. The left end-point of an interval I is denoted by $l(I)$ and the right end-point, for bounded I, is denoted by $r(I)$. An interval I is *singular* iff it is of the form $[a,a]$; that is, I is closed and $l(I) = r(I)$. Two intervals I and I' are *adjacent* iff (1) either I is right-open and I' is left-closed, or I is right-closed and I' is left-open, and (2) $r(I) = l(I')$. For instance, the intervals $(1,2]$ and $(2,2.5)$ are adjacent.

An *interval sequence* $\overline{I} = I_0 I_1 I_2 \ldots$ is a finite or infinite sequence of intervals that partitions the real line:

1. Any two neighboring intervals I_i and I_{i+1} are adjacent.

2. For all $t \in \mathrm{R}$, there is some interval I_i with $t \in I_i$.

In particular, I_0 is left-closed and $l(I_0) = 0$. The last interval of any finite interval sequence is unbounded.

Real-time systems

A *real-time system* $\mathrm{S} = (\mathcal{S}, \mathcal{P}, \mu, \mathcal{T})$ consists of four components:

- A set $\mathcal{S}$ of system *states*.

- A set $\mathcal{P}$ of *observables*. The set $\mathcal{P}$ typically contains either observable events or observable propositions about the system state. Each element in the powerset $2^{\mathcal{P}}$ is a possible *observation*.

- A *labeling function* μ from $\mathcal{S}$ to $2^{\mathcal{P}}$ that determines the observable component of each system state. If $\mathcal{P}$ consists of events, then every state $s \in \mathcal{S}$ is labeled with the (possibly empty) set $\mu(s) \subseteq \mathcal{P}$ of events that are observed when the system is in state s. If $\mathcal{P}$ consists of propositions, then s is labeled with the set $\mu(s)$ of observable propositions that are true in state s.

- A fusion-closed set $\mathcal{T}$ of *timed state sequences*.

  Each timed state sequence $\tau \in \mathcal{T}$ represents a system behavior by identifying a unique system state $\tau(t) \in \mathcal{S}$ with every time instant $t \in \mathrm{R}$. Formally, a timed state sequence τ is a function from R to $\mathcal{S}$ that satisfies the *finite-variability* condition:

  > There exists an interval sequence $\overline{I} = I_0 I_1 I_2 \ldots$ such that throughout each interval I_i, the observable component of the system state does not change; that is, $\mu(\tau(t)) = \mu(\tau(t'))$ for all $t, t' \in I_i$. The interval sequence $\overline{I}$ is called *compatible* with the timed state sequence τ.

  Instantaneous events correspond to singular intervals. The finite-variability assumption asserts that in any bounded interval of time, there can be only finitely many observable events or state changes.

  A set $\mathcal{T}$ of timed state sequences is *fusion-closed* if each system state contains all information necessary to determine the future evolution of the system:

  > For all timed state sequences $\tau_1, \tau_2 \in \mathcal{T}$ and time instants $t_1, t_2 \in \mathrm{R}$, if $\tau_1(t_1) = \tau_2(t_2)$, then $\tau \in \mathcal{T}$ for the timed state sequence τ with $\tau(t) = \tau_1(t)$ for $t \leq t_1$ and $\tau(t) = \tau_2(t + t_2 - t_1)$ for $t > t_1$.

As a concrete example, let us consider a controller that responds to an environment stimulus. Whenever the stimulus p occurs, the controller reaches, within one time unit, an internal decision to either perform the response q two time units after the stimulus or not to respond at all. If a new stimulus occurs before the previous one has been served, by a response or by the decision not to respond, it is ignored. Suppose that S is the real-time system consisting of the environment and the controller. The only observables $\mathcal{P} = \{p, q\}$ are the environment stimulus p and the system response q. We assume that both events, p and q, are instantaneous. A state of the system must contain enough information to determine the future evolution of the system. Thus, we may characterize the states $\mathcal{S}$ of S as triples of the form (s_0, s_1, s_2):

- The boolean parameter s_0 indicates if there is a current stimulus.

- The ternary parameter $s_1 \in \{\mathit{none}, \mathit{respond}, \mathit{ignore}\}$ indicates which decision has been reached concerning the most recent stimulus that has not been ignored.

- The real parameter $s_2 \in [0, 2] \cup \{\infty\}$ indicates the time that has elapsed since the most recent stimulus that has not been ignored. In particular, $s_2 = \infty$ if more than 2 time units have elapsed.

The labeling function μ is defined as follows:

$p \in \mu(s_0, s_1, s_2)$ iff $s_0 = \mathit{true}$;
$q \in \mu(s_0, s_1, s_2)$ iff $s_1 = \mathit{respond}$ and $s_2 = 2$.

The following timed state sequence $\hat{\tau}$ describes one possible behavior of the system S:

$\hat{\tau}(t) = (\mathit{false}, \mathit{none}, \infty)$ for $t \in [0, 1)$,
$\hat{\tau}(1) = (\mathit{true}, \mathit{none}, 0)$,
$\hat{\tau}(t) = (\mathit{false}, \mathit{none}, t-1)$ for $t \in (1, 1.5)$,
$\hat{\tau}(t) = (\mathit{false}, \mathit{respond}, t-1)$ for $t \in [1.5, 3]$,
$\hat{\tau}(t) = (\mathit{false}, \mathit{respond}, \infty)$ for $t > 3$.

According to this behavior, a single stimulus occurs, at time 1. At time 1.5 the system decides to respond and does so at time 3.

Observation sequences

An *observation sequence* $\overline{\sigma} = \sigma_0\sigma_1\sigma_2\ldots$ is a finite or infinite sequence of observations $\sigma_i \subseteq 2^{\mathcal{P}}$. A *timed observation sequence* $\rho = (\overline{\sigma}, \overline{I})$, also written as a sequence

$$(\sigma_0, I_0) \rightarrow (\sigma_1, I_1) \rightarrow (\sigma_2, I_2) \rightarrow \cdots$$

of observation-interval pairs, consists of an observation sequence $\overline{\sigma}$ and an interval sequence $\overline{I}$ of equal length. The timed observation sequence ρ is *stutter-free* iff any two neighboring observations σ_i and σ_{i+1} are distinct. A set of timed observation sequences is closed under *stuttering* iff whenever the set contains a timed observation sequence of the form

$$\cdots \rightarrow (\sigma_i, I_i) \rightarrow \cdots$$

and $I_i = I_i' \cup I_i''$ for adjacent intervals I_i' and I_i'', then it contains also the timed observation sequence

$$\cdots \rightarrow (\sigma_i, I_i') \rightarrow (\sigma_i, I_i'') \rightarrow \cdots.$$

The stuttering closure of a timed observation sequence ρ is the smallest set of timed observation sequences that contains ρ and is closed under stuttering.

From a timed state sequence τ, we can extract a set $\mu(\tau)$ of timed observation sequences that represent the observations made when the system exhibits the behavior described by τ. Formally, a timed observation sequence $(\overline{\sigma}, \overline{I})$ is contained in $\mu(\tau)$ iff the interval sequence $\overline{I}$ is compatible with τ and $\sigma_i = \mu(\tau(t))$ for $t \in I_i$. It is easy to check that the set $\mu(\tau)$ is the stuttering closure of a single stutter-free timed observation sequence. For instance, the timed state sequence $\hat{\tau}$ of the example given above yields the stuttering closure $\mu(\hat{\tau})$ of the timed observation sequence

$$(\{\}, [0, 1)) \rightarrow (\{p\}, [1, 1]) \rightarrow (\{\}, (1, 3)) \rightarrow (\{q\}, [3, 3]) \rightarrow (\{\}, (3, \infty)).$$

Special cases of real-time systems

If we restrict ourselves to the analysis of certain systems, we may assume that all interval sequences are of a particular form. Here we present some common assumptions about real-time systems that lead to one or both of the following simplifications of interval sequences:

1. If the interval types in a sequence follow a particular pattern, interval sequences can be represented as sequences of time instants. Suppose, for instance, that we observe only instantaneous events. Then we may restrict ourselves to timed observation sequences containing singular intervals, associated with events, that alternate with open intervals, indicating the distance between events. In this case, it suffices only to record the time instants at which

events occur. In our example, the observable component of the timed state sequence $\hat{\tau}$ is completely specified by the sequence

$$(\{p\}, 1) \rightarrow (\{q\}, 3).$$

Similarly, suppose that we observe only propositions about the system states that have a duration in time, and suppose that whenever there is a change in the observation from state s to state s', the observation at the transition point is $\mu(s')$. Then we may limit ourselves to timed observation sequences all of whose intervals are left-closed and right-open. In this case, it suffices to record the left end-points of all intervals. For instance, the sequence $0, 3, 3.5$ of real numbers could stand for the interval sequence $[0, 3), [3, 3.5), [3.5, \infty)$. In the symmetric case, we may restrict ourselves to timed observation sequences all of whose intervals are left-open and right-closed.

2. If all events or state changes occur with the ticks of a global clock, the nonnegative integers N suffice as time domain. We call such systems *synchronous*.

2.2 Abstract semantics

Given our concrete semantics of a real-time system as a set of timed state sequences, one may choose several abstractions to model systems by simpler objects than sets of timed state sequences. The system aspects in which we are interested determine the abstraction mechanisms we can employ. We discuss three common abstractions, which are independent of each other — first, the restriction only to observable linear behaviors; second, the linearization of simultaneous activities; and, third, the digitization of time. The paradigm of digitization, which is obviously related to timing considerations, is the only one of these three abstraction mechanisms that is particular to the modeling of real-time systems.

A Trace semantics

Trace semantics sacrifices information about the internal structure of a system. If we are content to analyze the observable traces of a system, we may ignore all nonobservable state components. Thus, trace semantics identifies a real-time system with the set of timed observation sequences that are obtained from the timed state sequences representing possible system behaviors. The trace semantics of the system $\mathrm{S} = (\mathcal{S}, \mathcal{P}, \mu, \mathcal{T})$ is the pair $(\mathcal{P}, \mathcal{R})$, where the set $\mathcal{R}$ of timed observation sequences is the union of all sets $\mu(\tau)$ for $\tau \in \mathcal{T}$. For instance, the trace semantics of our stimulus-response example contains all timed observation sequences in which (1) the difference between any two system responses q is at least two time units and (2) every response q is preceded by an environment stimulus p two time units earlier.

B Interleaving semantics

Interleaving semantics sacrifices information about the simultaneity of activities. Independent simultaneous events are often nondeterministically sequentialized so that at most one action of a distributed system has to be analyzed at any point in a timed state sequence (or a timed observation sequence). To allow the interleaving of simultaneous activities in our framework, we need to generalize our definitions concerning interval sequences:

- Two intervals I and I' are *almost* adjacent iff $r(I) = l(I')$; that is, they have at most one point in common. If we relax the requirement of adjacency of neighboring intervals in an interval sequence to almost adjacency, we obtain the notion of *weakly-monotonic* interval sequences. An example of a weakly-monotonic interval sequence is the sequence

$$[0, 1.3], [1.3, 1.3], [1.3, 4], [4, 5], [5, \infty).$$

- A *weakly*-timed observation sequence consists, then, of an observation sequence and a corresponding weakly-monotonic interval sequence.

- A weakly-timed state sequence maps every time instant $t \in \mathrm{R}$ to a finite sequence of system states. The number of states in this sequence is called the multiplicity m_t of t. Formally, a *weakly*-timed state sequence τ associates a multiplicity m_t with every time instant $t \in \mathrm{R}$ and maps every pair (t, i), for $t \in \mathrm{R}$ and $1 \leq i \leq m_t$, to a system state $\tau(t, i) \in \mathcal{S}$. In addition, the mapping τ is finitely variable; that is, there exists a weakly-timed observation sequence $(\overline{\sigma}, \overline{I})$ and a monotone onto mapping from lexicographic pairs (t, i) to indexes $j \geq 0$ such that (1) $t \in I_j$ and (2) $\mu(\tau(t, i)) = \sigma_j$. The lexicographically ordered pairs (t, i) can be viewed as specifying a metric "macro-time" $t \in \mathrm{R}$ and a linearly ordered discrete "micro-time" $1 \leq i \leq m_t$.

Using this generalized notion of timed state sequences, any number of simultaneous events can be modeled as a sequence of events at the same time instant. Consider, for example, the timed observation sequence

$$(\{\}, [0, 3)) \rightarrow (\{p, q\}, [3, 3]) \rightarrow (\{\}, (3, \infty)),$$

in which the two events p and q occur at time 3. An interleaving semantics may model this behavior, instead, by the two weakly-timed observation sequences

$$(\{\}, [0, 3)) \rightarrow (\{p\}, [3, 3]) \rightarrow (\{q\}, [3, 3]) \rightarrow (\{\}, (3, \infty)),$$

$$(\{\}, [0, 3)) \rightarrow (\{q\}, [3, 3]) \rightarrow (\{p\}, [3, 3]) \rightarrow (\{\}, (3, \infty)).$$

Indeed, an interleaving semantics typically abstracts further by assuming that all observation intervals are either closed or unbounded. The behavior of a system is viewed as a two-phase activity [HMP90]: macro-phases, during which time (i.e., macro-time) advances, alternate with micro-phases, during which the observable part of the system state changes finitely often. The micro-phases result from the interleaving of independent instantaneous events and correspond to finite repetitions of singular intervals. Under these assumptions, weakly-monotonic interval sequences may be represented by sequences of weakly monotonically increasing real numbers. For instance, the behavior of our example is often modeled by the two sequences

$$(\{\}, 0) \rightarrow (\{\}, 3) \rightarrow (\{p\}, 3) \rightarrow (\{q\}, 3) \rightarrow (\{\}, 3),$$

$$(\{\}, 0) \rightarrow (\{\}, 3) \rightarrow (\{q\}, 3) \rightarrow (\{p\}, 3) \rightarrow (\{\}, 3),$$

each of which contains one micro-phase, at time 3, between two macro-phases during the intervals $[0, 3]$ and $[3, \infty)$, respectively.

C Fictitious-clock semantics

Fictitious-clock semantics sacrifices information about the precise times of activities. The introduction of a global fictitious clock, which records the times of events with finite precision only, allows the use of the nonnegative integers as time domain, thereby simplifying the reasoning about time. If several events fall between two clock ticks, then the fictitious clock can distinguish them only by temporal ordering, not by time.

To allow the digitization of times, we have to consider, once again, weakly-monotonic interval sequences. Using the notion of weakly-timed state sequences, any number of events that occur between two ticks of the fictitious clock can be modeled as a sequence of events at the "same" integer time. Consider, for example, the timed observation sequence

$$(\{p\},[0,1.3)) \to (\{q\},[1.3,1.3]) \to (\{p\},(1.3,1.8]) \to (\{q\},(1.8,\infty))$$

of a system that satisfies the proposition q at time 1.3 and after time 1.8, and otherwise it satisfies the proposition p. We assume that a fictitious clock ticks at all integer times. The fictitious-clock semantics may model this behavior, then, by the weakly-timed observation sequence

$$(\{p\},[0,1]) \to (\{q\},[1,1]) \to (\{p\},[1,1]) \to (\{q\},[1,\infty)),$$

which results from ignoring the fractional parts of all interval end-points.

Summary

Any combination of system restrictions and choices of abstractions we presented leads to one of sixteen possible formal semantics of real-time systems:

1. *State sequences or observation sequences.* Observation sequences suffice for the study of the observable traces of a system.

2. *Time intervals or time points.* Time points suffice in a variety of cases, such as the study of instantaneous events and under the assumption of two-phase interleaving.

3. *Strictly monotonic or weakly monotonic time.* Time is weakly monotonic if adjacent time intervals may overlap in a common point, or if adjacent time instants may be identical. Weakly monotonic time is necessary for employing one or both of two independent abstractions — the modeling of simultaneous activities by interleaving and the recording of times by a fictitious clock.

4. *Real-numbered time or integer time.* Integer time suffices for the study of synchronous systems and in the presence of a fictitious clock.

In previous papers, we investigated the following combinations of semantical options:

- an observation-oriented point-based weakly-monotonic integer-time semantics [AH89, AH90, Hen90, HMP91];

- an observation-oriented point-based weakly-monotonic real-time semantics [Hen91b];

- an observation-oriented point-based strictly-monotonic real-time semantics [AD90];

- an observation-oriented interval-based strictly-monotonic real-time semantics [AFH91, AH91];

- a state-oriented point-based strictly-monotonic real-time semantics [ACD90];
- a state-oriented interval-based strictly-monotonic real-time semantics [Alu91].

In this paper, we review our results and identify the semantical assumptions that were necessary to obtain them. This allows us to generalize many results to a broader spectrum of semantical choices than was previously known.

3 Real-time Logics

We discuss three ways of extending the syntax of temporal logic for specifying real-time systems. The main point of this section is to demonstrate that these extensions apply equally to any particular choice of temporal logic.

3.1 Choosing the temporal logic

Temporal logic (or tense logic, the term sometimes preferred by philosophers) is the class designation for modal logics whose modal operators are interpreted in a temporal manner: the basic operator $\Box$ is interpreted as "always" and, consequently, its dual $\Diamond$ means "eventually." The use of temporal logic as a formalism for specifying the behavior of reactive systems over time was first proposed by Pnueli [Pnu77] and has been studied extensively since then. Temporal logic provides a succinct and natural way of expressing the desired *qualitative* temporal requirements of speed-independent systems, including invariance, precedence, and responsiveness (cf. [Pnu86]). The traditional temporal operators, however, cannot refer to metric time and, hence, are insufficient for the specification of *quantitative* temporal requirements, or so-called *hard* real-time constraints, which put timing deadlines on the behavior of reactive systems.

Having emphasized that the term temporal logic covers a variety of particular logical formalisms, with syntactic as well as semantic differences, we refer the reader to [Eme90] for a comprehensive overview of individual temporal logics that have been used for specifying and verifying reactive systems. Any of these logics can be, and some have been, extended by the constructs we will present for expressing timing constraints. Before proceeding to the issue of real time, however, let us briefly review some of the design decisions involved in choosing a particular underlying logic for qualitative temporal reasoning.

First, depending on the nature of the problems we wish to formalize, we use either a first-order temporal logic or the propositional fragment. This choice is determined by the data domains over which the reactive systems under investigation operate, and interests us little, as we treat the issue of time independent of other system parameters. We are mostly concerned with propositional fragments, because those allow us to study issues of expressiveness and complexity without interference by data considerations.

Second, many of the temporal logics that are used in computer science can be classified into linear-time and branching-time logics. This distinction is important, because it determines if we can choose a trace semantics for real-time systems. Throughout this section, we assume a given real-time system $\mathrm{S} = (\mathcal{S}, \mathcal{P}, \mu, \mathcal{T})$ with the trace semantics $(\mathcal{P}, \mathcal{R})$. The set $\mathcal{P}$ of observables constitutes the set of atomic propositions for the logics we discuss. Hence, the truth value of any atomic proposition is fully determined by the observable component of a system state ($s \models p$ iff $p \in \mu(s)$).

1. *Linear-time* logics are interpreted over linear structures of states. Every state sequence represents an execution sequence of a reactive system. A classical example of a linear-time logic

is PTL [GPSS80]. In PTL, the typical response property that "every environment stimulus p must be followed by a system response q" is defined by the formula

$$\Box(p \rightarrow \Diamond q),$$

which requires that in any possible behavior, if the system is in a state in which p is observed, then it will, at some later point, be in a state in which q is observed. Real-time extensions of PTL introduce a way of defining timing requirements such as the time-bounded response property that "every stimulus p is followed by a response q within 3 time units."

Formally, the real-time system S satisfies a linear-time formula ϕ iff every timed state sequence in $\mathcal{T}$ satisfies ϕ. Since the truth value of ϕ over a timed state sequence τ is completely determined by the observable component of τ, it suffices to consider the trace semantics of S and interpret ϕ over timed observation sequences: S satisfies ϕ iff $\rho \models \phi$ for every timed observation sequence $\rho \in \mathcal{R}$.

2. *Branching-time* temporal logics, on the other hand, are interpreted over tree structures of states. Every tree represents a reactive system, whose possible execution sequences correspond to the paths in the tree. Classical examples of branching-time logics include $\mathcal{UB}$ [BMP81], CTL [EC82], and CTL* [CES86]. In CTL, the property that "every stimulus p is possibly followed by a response q" is defined by the formula

$$\forall\Box(p \rightarrow \exists\Diamond q),$$

which asserts that in any possible behavior, if the system is in a state in which p is observed, then there is a possible continuation along which the system will, at some later point, be in a state in which q is observed. Real-time extensions of CTL allow to put a time bound on the distance between the stimulus p and the response q.

Since we require that each state $s \in \mathcal{S}$ contains all the information necessary to decide the future behavior of the real-time system S, the set $\mathcal{T}$ of timed state sequences contains all the branching information for constructing a unique tree, with root s, whose paths represent the possible behaviors of S if started in state s. This is because the formal requirement of fusion closure for $\mathcal{T}$ captures the intuitive notion of a tree; it ensures that the reachability relation over $\mathcal{S}$ induced by $\mathcal{T}$ is transitive. Thus, the truth value $s \models \phi$ of a branching-time formula ϕ can be determined for each state $s \in \mathcal{S}$. The real-time system S satisfies ϕ iff $s \models \phi$ for all $s \in \mathcal{S}$.

It follows that linear-time logics employ an observation-oriented semantics, and branching-time logics employ a state-oriented semantics.

Third, there is the choice of temporal operators. The temporal operator most typically employed is the *until* operator $\mathcal{U}$, which can be used to define the *always* operator $\Box$ and provides a generally desired level of expressiveness [GPSS80]. Many temporal logics exclude the *next* operator to ensure that the models of a formula are closed under stuttering [Lam83]. Some logics include past temporal operators such as *since*, the dual of *until* [LPZ85]. Branching-time formulas are built from linear-time formulas using path quantifiers. The logic CTL constrains the linear-time formulas that can be used to a very simple form by coupling the path quantifiers with the temporal operators. The logic CTL* allows arbitrary linear-time formulas.

3.2 Writing the timing constraints

The fourth question raised when defining the syntax of a real-time extension for a temporal logic is how to incorporate timing requirements in a formula. We consider three possible solutions.

A Bounded temporal operators

A common way of introducing real time in the syntax replaces the unrestricted temporal operators by time-bounded versions. For example, the bounded operator $\Diamond_{[2,4]}$ is interpreted as "eventually within 2 to 4 time units." This approach to the specification of timing properties has been advocated by Koymans, Vytopil, and de Roever [KVdR83, KdR85, Koy90], although an early proposal by Bernstein and Harter can be viewed as a precursor [BH81]. More applications of the bounded-operator method for expressing timing constraints can be found in [SPE84, Har88, PH88, HW89, Lew90]. Bounded operators have been analyzed for their expressiveness and complexity in [EMSS89, ACD90, AH90, AFH91].

To be concrete, let us define a propositional linear-time logic that employs time-bounded temporal operators. The formulas of this bounded-operator logic are built from atomic propositions by boolean connectives and time-bounded versions of the *until* operator. The *until* operator may be bounded (i.e., subscripted) by any interval with rational end-points. Hence, the bounded-operator formulas are inductively defined as follows:

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

where $p \in \mathcal{P}$ and the end-points of the interval I are rational numbers.

We interpret the formulas of this linear-time logic over timed observation sequences, which provide a unique interpretation for the atomic propositions at every time instant. Informally, the formula $\phi_1 \mathcal{U}_I \phi_2$ holds at time t in a timed observation sequence iff there is a later time instant $t' \in t + I$ such that ϕ_2 holds at time t' and ϕ_1 holds throughout the interval (t, t'). Formally, given a bounded-operator formula ϕ, a timed observation sequence $\rho = (\overline{\sigma}, \overline{I})$, and a time instant $t \in \mathrm{R}$, the satisfaction relation $(\rho, t) \models \phi$ is defined inductively as follows:

$(\rho, t) \models p$ iff $p \in \sigma_i$, where $t \in I_i$;
$(\rho, t) \models \neg\phi$ iff $(\rho, t) \not\models \phi$;
$(\rho, t) \models (\phi_1 \wedge \phi_2)$ iff $(\rho, t) \models \phi_1$ and $(\rho, t) \models \phi_2$;
$(\rho, t) \models \phi_1 \mathcal{U}_I \phi_2$ iff $(\rho, t') \models \phi_2$ for some $t' \in t + I$, and $(\rho, t'') \models \phi_1$ for all $t < t'' < t'$.

The timed observation sequence ρ satisfies the formula ϕ iff $(\rho, 0) \models \phi$.

Now we can introduce 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 $\mathit{true}\, \mathcal{U}_I \phi$ and $\neg\Diamond_I \neg\phi$, respectively. It follows that the formula $\Box_I \phi$ (or $\Diamond_I \phi$) holds at time $t \in \mathrm{R}$ of a timed observation sequence iff ϕ holds at all times (at some time, respectively) within the interval $t + I$. The typical time-bounded response property that "every stimulus p is followed by a response q within 3 time units" may then be defined by the bounded-operator formula

$$\Box(p \rightarrow \Diamond_{[0,3]} q),$$

which is usually written as

$$\Box(p \rightarrow \Diamond_{\leq 3} q). \qquad (\dagger)$$

We have chosen an interval-based strictly-monotonic real-time semantics for our sample bounded-operator logic. It is not difficult to interpret the same set of formulas over alternative semantics. For instance, we may adopt an interleaving semantics and interpret bounded-operator formulas over weakly-timed observation sequences [Hen91b].

B Freeze quantification

The bounded-operator notation can relate only adjacent temporal contexts. Consider, for instance, the property that "every stimulus p is followed by a response q and, then, by another response r such that r is within 5 time units of the stimulus p." There is no direct way of expressing this "nonlocal" timing requirement using time-bounded operators. This shortcoming of bounded temporal operators can be remedied by extending temporal logic with explicit references to the times of temporal contexts. We discuss two such methods. In this subsection, we access the time of a state through a quantifier, which binds ("freezes") a variable to the corresponding time; in the next subsection, we access the time of a state through a (dynamic) state variable.

The idea of *freeze* quantification was introduced and has been analyzed by the authors [AH89, Hen90, Alu91, Hen91b]. We present it here by considering again the propositional linear-time case. The freeze quantifier "$x.$" binds the associated variable x to the time of the current temporal context: the formula $x.\,\phi(x)$ holds at time t iff $\phi(t)$ does. Thus, in the formula $\Diamond y.\,\phi$, the time variable y is bound to the time of the state at which ϕ is "eventually" true. By admitting atomic formulas that relate the times of different states, we can write the time-bounded response property (†) as

$$\Box x.\,(p \rightarrow \Diamond y.\,(q \wedge y \leq x + 3)).$$

We read the this formula as "in every state with time x, if p holds, then there is a later state with time y such that q holds and y is at most $x + 3$." The nonlocal property that "every stimulus p is followed by a response q and, then, by another response r within 5 time units of the stimulus p" may be specified by the formula

$$\Box x.\,(p \rightarrow \Diamond(q \wedge \Diamond z.\,(r \wedge z \leq x + 5))). \qquad (‡)$$

Freeze quantification allows only references to times that are associated with states. Consequently, the freeze quantifier $x.$ behaves differently from standard first-order quantifiers over time; it is, for example, its own dual:

$$\neg(x.\,\phi) \leftrightarrow x.\,(\neg\phi).$$

Since the expressive power of a modal logic with freeze quantification lies, in general, between the expressive power of the corresponding propositional and first-order modal logics, we refer to a logic with freeze quantification as *half-order* [Hen90].

Let us now be more precise and define a half-order linear-time logic. Given a set V of time variables and a set $\Pi(V)$ of atomic timing constraints with free variables from V, the half-order formulas are inductively defined as follows:

$$\phi := p \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U} \phi_2 \mid x.\,\phi$$

for $x \in V$, $p \in \mathcal{P}$, and $\pi \in \Pi(V)$. Additional temporal operators such as $\Diamond$ (*eventually*) and $\Box$ (*always*) can be defined in terms of the *until* operator as usual. The atomic timing constraints (e.g., $y \leq x + 3$) typically involve comparisons between terms containing variables, constants, and primitive operations on time such as addition.

We interpret the half-order formulas again over timed observation sequences (other semantical choices are, of course, possible). The timed observation sequence $\rho = (\overline{\sigma}, \overline{I})$ satisfies the formula ϕ iff $(\rho, 0) \models_{\mathcal{E}} \phi$ for every environment $\mathcal{E}\colon V \rightarrow \mathrm{R}$, where the satisfaction predicate $\models$ is inductively defined as follows:

$(\rho, t) \models_{\mathcal{E}} p$ iff $p \in \sigma_i$, where $t \in I_i$;
$(\rho, t) \models_{\mathcal{E}} \pi$ iff $\mathcal{E} \models \pi$;
$(\rho, t) \models_{\mathcal{E}} \neg\phi$ iff $(\rho, t) \not\models_{\mathcal{E}} \phi$;
$(\rho, t) \models_{\mathcal{E}} \phi_1 \wedge \phi_2$ iff $(\rho, t) \models_{\mathcal{E}} \phi_1$ and $(\rho, t) \models_{\mathcal{E}} \phi_2$;
$(\rho, t) \models_{\mathcal{E}} \phi_1 \mathcal{U} \phi_2$ iff $(\rho, t') \models_{\mathcal{E}} \phi_2$ for some $t' \in t+I$, and $(\rho, t'') \models_{\mathcal{E}} \phi_1$ for all $t < t'' < t'$;
$(\rho, t) \models_{\mathcal{E}} x.\phi$ iff $(\rho, t) \models_{\mathcal{E}[x:=t]} \phi$.

Here $\mathcal{E}[x := t]$ denotes the environment that agrees with the environment $\mathcal{E}$ on all variables except x, which is mapped to $t \in \mathrm{R}$.

If the atomic timing constraints in $\Pi(V)$ permit at least comparisons and addition of constants, then the bounded-operator notation is a fragment of the freeze-quantifier notation: the bounded-operator formula $\phi_1 \mathcal{U}_I \phi_2$ is equivalent to the half-order formula

$$x.(\phi_1 \mathcal{U} y.(y \in x + I \wedge \phi_2)).$$

C Explicit clock variable

A third way of writing real-time requirements is based on standard first-order temporal logic. The syntax uses a dynamic state variable T — the *clock* variable — and first-order quantification for global (rigid) variables over the time domain. The clock variable T assumes, in each state, the value of the corresponding time. For instance, the time-bounded response property (†) can be specified by the formula

$$\forall x.\Box((p \wedge T = x) \rightarrow \Diamond(q \wedge T \leq x + 3)).$$

Here, the global variable x is bound to the time of every state in which p is observed. We refer to the use of a clock variable as the "explicit-clock" notation. Examples of this method for expressing timing constraints can be found in [PdR82, Ron84, Har88, PH88, Ost90, LA]; it has been studied for its expressiveness and complexity in [AH90, HLP90].

Let us once again define a propositional linear-time logic with an interval-based strictly-monotonic real-time semantics. As before, let V be a set of (global) time variables, and let $\Pi(V \cup \{T\})$ be a set of timing constraints over the variables in $V \cup \{T\}$. The explicit-clock formulas are inductively defined as follows:

$$\phi := p \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U} \phi_2 \mid \exists x.\phi$$

for $x \in V$, $p \in \mathcal{P}$, and $\pi \in \Pi(V \cup \{T\})$. The timed observation sequence $\rho = (\overline{\sigma}, \overline{I})$ satisfies the formula ϕ iff $(\rho, 0) \models_{\mathcal{E}} \phi$ for every environment $\mathcal{E} : V \rightarrow \mathrm{R}$. Only the following two clauses in the definition of the satisfaction predicate $\models$ differ from the evaluation of half-order formulas:

$(\rho, t) \models_{\mathcal{E}} \pi$ iff $\mathcal{E}[T := t] \models \pi$;
$(\rho, t) \models_{\mathcal{E}} \exists x.\phi$ iff $(\rho, t) \models_{\mathcal{E}[x:=t']} \phi$ for some $t' \in \mathrm{R}$.

Assuming the same set of atomic timing constraints, the half-order notation is a fragment of the explicit-clock notation: the half-order formula $x.\phi$ is equivalent to the first-order formula

$$\exists x.(T = x \wedge \phi)$$

(or, alternatively, to the formula $\forall x.(T = x \rightarrow \phi)$).

3.3 Examples of real-time logics

After considering the various choices available for defining a real-time temporal logic, let us examine the decisions that have been made for some logics proposed in the literature. To summarize the discussion so far, we list the questions that need to be answered when designing a real-time temporal logic:

1. Is it propositional or first-order?
2. Is it linear-time or branching-time?
3. Which temporal operators are used?
4. Which semantic abstractions such as interleaving or a fictitious clock are assumed?
5. Of the three possible choices for writing timing constraints — bounded operators, freeze quantification, or a clock variable — which one is used?
6. What are the primitive operations of the assertion language for timing constraints?

We emphasize that all of the above choices are independent.

Linear-time logics

The logics MTL (*metric* temporal logic) [AH90], TPTL (*timed* temporal logic) [AH89], RTTL (*real-time* temporal logic) [Ost90], and XCTL (for *explicit-clock* temporal logic) [HLP90] are linear-time logics that assume both the interleaving and fictitious-clock abstractions; that is, they are defined over a point-based weakly-monotonic integer-time semantics:

- MTL is a propositional bounded-operator logic; its temporal operators include time-bounded versions of the *until*, *next*, *since*, and *previous* (the past dual of *next*) operators.
- TPTL is a propositional half-order logic that uses only the future temporal operators *until* and *next*; its atomic timing constraints include the primitives $\leq$ (comparison), $\equiv_c$ (congruence modulo a constant), and $+c$ (addition by an integer constant).
- RTTL is a first-order explicit-clock logic. Although RTTL was defined without restrictions on the assertion language for atomic timing constraints, we shall refer by the name RTTL to the propositional fragment with the primitives of TPTL for timing constraints.
- XCTL is a propositional explicit-clock logic, whose assertion language for atomic timing constraints allows the primitives of comparisons and addition. Thus, the timing constraints of XCTL are richer than those of the previous logics, which prohibit the addition of time variables. XCTL, on the other hand, forbids explicit quantification over the time variables; that is, all global time variables are assumed to be implicitly universally quantified on the outside of any given formula.

While all of the above logics assume integer time, the logic MITL (*metric interval* temporal logic) [AFH91] employs the nonnegative reals as time domain. MITL is a propositional linear-time logic with an interval-based strictly-monotonic real-time semantics; that is, it is interpreted over timed observation sequences, just like our three sample logics. MITL uses the bounded-operator syntax with the restriction that the temporal operators must not be bounded (subscripted) by singular intervals. For example, the formula

$$\Box(p \rightarrow \Diamond_{[3,3]}\, q)$$

is disallowed; that is, MITL rules out a form of equality constraints.

Branching-time logics

The logic RTCTL (for *real-time* computation tree logic) [EMSS89] is a propositional branching-time logic for synchronous systems; it is a bounded-operator extension of CTL with a point-based strictly-monotonic integer-time semantics.

The logic TCTL (*timed* computation tree logic) [ACD90] is a propositional branching-time logic with a less restrictive semantics; it is a bounded-operator extension of CTL with a point-based strictly-monotonic real-time semantics. A later version of the logic uses an interval-based semantics and half-order syntax with the primitives of comparisons and addition by constants [Alu91].

4 Automata-based Real-time Formalisms

We discuss an extension of finite automata and an extension of transition systems to specify real-time systems.

4.1 Timed automata

Timed automata were proposed as an abstract model for real-time systems [Dil89, AD90] (see also the article *The Theory of Timed Automata* in this volume). The formalism of timed automata generalizes finite-state machines over infinite strings. While ω-automata generate (or accept) infinite sequences of states (cf. [Tho90]), timed automata are additionally constrained by timing requirements and produce timed state sequences. While timed automata were originally given a point-based strictly-monotonic real-time semantics, we present an interval-based variant [AFH91]. It is also not difficult to interpret timed automata under the semantic assumptions of interleaving, a fictitious clock, and/or synchronicity.

A timed automaton operates with finite control — a finite set of locations and a finite set of real-valued clocks. All clocks proceed at the same rate and measure the amount of time that has elapsed since they were started (or reset). Each edge of the automaton may reset some of the clocks; each location of the automaton puts certain constraints on the values of the atomic propositions as well as on the values of the clocks: the control of the automaton can reside in a particular location only if the values of the propositions and clocks satisfy the corresponding constraints.

Formally, a *timed automaton* $\mathrm{A} = (\mathcal{P}, Q, Q_0, C, \mu, \nu, E, F)$ consists of eight components:

- A set $\mathcal{P}$ of *propositions*.
- A finite set Q of *locations*.
- A subset $Q_0 \subseteq Q$ of *initial locations*.
- A finite set C of *clocks*.
- A labeling function μ that assigns to each location in Q a boolean formula over the set $\mathcal{P}$ of propositions. For $\ell \in Q$, the formula $\mu(\ell)$ is called a *propositional constraint*.

- A labeling function ν that assigns to each location in Q a *timing constraint* over the variables in C. Each timing constraint is a boolean combination of atomic timing constraints from a set $\Pi(C)$, which typically contains comparisons between terms involving clock variables and primitive operations such as addition by constants. For $\ell \in Q$, the timing constraint $\mu(\ell)$ constrains the values of the clocks.

- A set $E \subseteq Q^2 \times 2^C$ of *edges*. Each edge (ℓ, ℓ', λ) identifies a source location ℓ, a target location ℓ', and a set $\lambda \subseteq C$ of clocks to be reset. The target location ℓ' is called an E-successor of the source location ℓ.

- A family $F \subseteq 2^Q$ of *acceptance sets* of locations. The (Muller) acceptance condition requires that the set of locations that is visited infinitely often during any run of the automaton A belongs to the acceptance family F.

The runs of a timed automaton define timed state sequences. At any time instant during a run, the configuration of the automaton is completely determined by the location in which the control resides and the values of all propositions and all clocks. The values of the clocks are given by a *clock interpretation* γ, which is a map from C to R: for any clock $x \in C$, the value of x under the interpretation γ is the nonnegative real number $\gamma(x)$. A *state* of the timed automaton A is a triple (ℓ, σ, γ), where $\ell \in Q$ is a location and

1. $\sigma \subseteq \mathcal{P}$ is an observation that satisfies the propositional constraint $\mu(\ell)$;

2. γ is a clock interpretation that satisfies the timing constraint $\nu(\ell)$.

Before defining the runs of A formally, let us give some intuition. Assume that, at time $t \in$ R, a timed automaton is in state (ℓ, σ, γ). Suppose that the location ℓ of the automaton and the observation σ remain unchanged during the time interval I with $l(I) = t$. All clocks proceed at the same rate as time elapses; at any time $t' \in I$ the value of any clock x is $\gamma(x) + t' - t$. During all this time, the clock values satisfy the timing constraint that is associated with ℓ:

$$(\gamma + t' - t) \models \nu(\ell).$$

Now suppose that the automaton changes its location at time $r(I) = t''$ via the edge (ℓ, ℓ', λ). This location change happens in one of two ways. If the interval I is right-closed, then the state at time t'' is $(\ell, \sigma, \gamma + t'' - t)$; otherwise, the state at time t'' is $(\ell', \sigma', \gamma')$, where σ' is an observation consistent with $\mu(\ell')$ and the clock interpretation γ' is defined by (1) $\gamma'(x) = 0$ for all clocks $x \in \lambda$, which are reset, and (2) $\gamma'(x) = \gamma(x) + t'' - t$ for all other clocks.

Let us formalize this intuition. A *run* of the automaton $\mathrm{A} = (\mathcal{P}, Q, Q_0, C, \mu, \nu, E, F)$ is a finite or infinite sequence

$$r: \quad \xrightarrow[\gamma_0]{} (\ell_0, \sigma_0, I_0) \xrightarrow[\gamma_1]{\lambda_1} (\ell_1, \sigma_1, I_1) \xrightarrow[\gamma_2]{\lambda_2} (\ell_2, \sigma_2, I_2) \xrightarrow[\gamma_3]{\lambda_3} \cdots$$

of locations $\ell_i \in Q$, observations $\sigma_i \subseteq \mathcal{P}$, intervals I_i, clock sets $\lambda_i \subseteq C$, and clock interpretations $\gamma_i : C \to$ R such that

1. $\ell_0 \in Q_0$;

2. $(s_i, s_{i+1}, \lambda_i) \in E$ for all $i \geq 0$;

3. σ_i satisfies $\mu(\ell_i)$ for all $i \geq 0$;

4. $\overline{I} = I_0 I_1 I_2 \ldots$ is an interval sequence;

5. for all $x \in C$ and $i \geq 0$, $\gamma_{i+1}(x) = 0$ if $x \in \lambda_{i+1}$, and $\gamma_{i+1}(x) = \gamma_i(x) + r(I_i) - l(I_i)$ otherwise;

6. $\gamma_i + t - l(I_i)$ satisfies $\nu(\ell_i)$ for all $i \geq 0$ and $t \in I_i$;

7. either $\overline{I} = I_0 I_1 \ldots I_n$ is finite and $\{\ell_n\} \in F$, or r is infinite and $\{\ell \mid \ell = \ell_i$ for infinitely many $i \geq 0\} \in F$.

Every run r of the timed automaton A uniquely determines a timed state sequence τ_r: let $\tau_r(t) = (\ell_i, \sigma_i, \gamma_i + t - l(I_i))$ for all $t \in I_i$. By $\mathcal{T}_{\mathrm{A}}$ we denote the set of all timed state sequences τ_r that correspond to runs of the automaton A. The timed automaton A defines, then, the real-time system $\mathrm{S}_{\mathrm{A}} = (\mathcal{S}, \mathcal{P}, \mu', \mathcal{T}_{\mathrm{A}})$, where $\mathcal{S}$ is the set of states of the automaton A and and the labeling function μ' is defined as the projection $\mu'(\ell, \sigma, \gamma) = \sigma$. It is easy to check that the set $\mathcal{T}_{\mathrm{A}}$ satisfies the requirement of fusion closure.

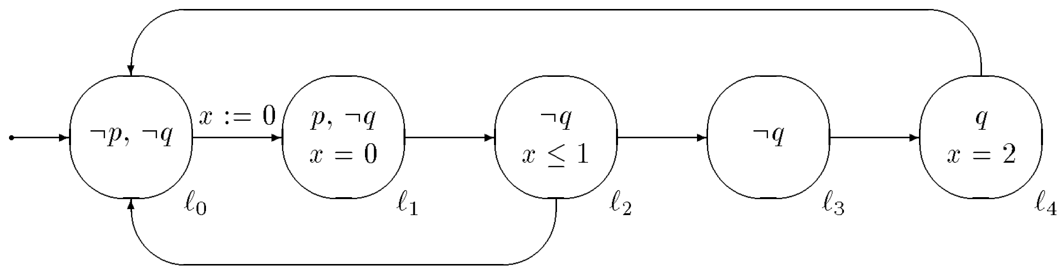
The set $\mu'(\tau_r)$ of timed observation sequences that are associated with the timed state sequence τ_r describes the observed behavior during the run r of the timed automaton A. Notice that the same stutter-free timed observation sequence may correspond to two different runs of A. This makes timed automata nondeterministic. The timed automaton A is called *deterministic* iff the following conditions hold for every location $\ell \in Q$:

1. For each pair $\ell_1, \ell_2 \in Q$ of E-successors of ℓ, either the propositional constraints $\mu(\ell_1)$ and $\mu(\ell_2)$ are mutually exclusive (i.e., contradictory), or the clock constraints $\nu(\ell_1)$ and $\nu(\ell_2)$ are mutually exclusive.

2. For each E-successor ℓ' of ℓ, either $\mu(\ell)$ and $\mu(\ell')$ are mutually exclusive, or $\nu(\ell)$ and $\nu(\ell')$ are mutually exclusive.

It is easy to check that for any deterministic timed automaton, there is a unique run to generate (or accept) a given timed observation sequence.

As an example, consider the following timed automaton A, which defines the real-time system we described informally in Section 2:



The automaton A has five locations and one clock, x; it starts in location ℓ_0 and every subset of locations containing the initial location ℓ_0 is an acceptance set. As soon as an external stimulus p occurs, the automaton A moves through location ℓ_1 to location ℓ_2. The clock x is used to measure the time that elapses from the time of the stimulus. Within one time unit after the stimulus, the automaton decides, nondeterministically, either to respond and proceed to location ℓ_3, or not to respond and return to location ℓ_0. If the decision is to respond, the automaton does so by issuing the instantaneous response q in location ℓ_4 two time units after the stimulus (note that all new stimuli that may have occurred in the meantime were ignored). Then the automaton returns to its start location ℓ_0, ready for the next stimulus, for which the clock x will be reused.

4.2 Timed transition systems

A different approach to the definition of real-time systems generalizes the formalism of transition systems [Kel76, Pnu77] by imposing timing constraints on the transitions [Ost90, Har88, LA90, HMP91] (see also the article *Timed Transition Systems* in this volume). A *transition system* $\mathrm{T}_0 = (\mathcal{P}, \mathcal{I}, \mathcal{E})$ consists of three components:

1. A set $\mathcal{P}$ of *propositions*. The subsets of $\mathcal{P}$, to which we refer as observations, are often called the "states" of the transition system. We refrain from this terminology to avoid confusion with our usage of the term "state."

2. A set $\mathcal{I} \subseteq 2^{\mathcal{P}}$ of *initial observations*.

3. A finite set $\mathcal{E}$ of *transitions*. Every transition $e \in \mathcal{E}$ is a binary relation on the set $2^{\mathcal{P}}$ of observations; that is, it defines for every observation $\sigma \subseteq \mathcal{P}$ a (possibly empty) set of e-successors $e(\sigma) \subseteq 2^{\mathcal{P}}$. A transition e is *enabled* on the observation σ iff $e(\sigma) \neq \emptyset$.

Time is incorporated into the transition system model, under the assumption that all transitions happen instantaneously, by restricting the times at which transitions may occur. The timing constraints on transitions are classified into two categories: lower-bound and upper-bound requirements. They ensure that transitions occur neither too early nor too late, respectively. Consequently, a *timed transition system* $\mathrm{T} = (\mathcal{P}, \mathcal{I}, \mathcal{E}, l, u)$ consists of an underlying transition system $\mathrm{T}_0 = (\mathcal{P}, \mathcal{I}, \mathcal{E})$ as well as

4. A *minimal delay* (lower bound) $l_e \in \mathrm{N}$ for each transition $e \in \mathcal{E}$.

5. A *maximal delay* (upper bound) $u_e \geq l_e$, with $u_e \in \mathrm{N} \cup \{\infty\}$, for each transition $e \in \mathcal{E}$.

Informally, the lower-bound requirement for a transition e asserts that e cannot be taken unless it has been continuously enabled for l_e time units. In the case that the maximal delay u_e is finite, the upper-bound requirement for the transition e asserts that e cannot be continuously enabled for more than u_e time units without being taken. An infinite maximal delay (i.e., $u_e = \infty$) puts a fairness condition on the transition e: it cannot be enabled perpetually without being taken. Formally, the timed transition system T defines the real-time system $\mathrm{S}_{\mathrm{T}} = (\mathcal{S}, \mathcal{P}, \mu, \mathcal{T})$:

- Every state $s \in \mathcal{S}$ consists of an observation $\mu(s) \subseteq \mathcal{P}$ and a delay $\delta_e(s) \in \mathrm{R}$ for each transition $e \in \mathcal{E}$:
$$\mathcal{S} = 2^{\mathcal{P}} \times \mathrm{R}^{\mathcal{E}}.$$
  The delay $\delta_e(s)$ indicates the time that has elapsed since the transition e became enabled.

- A timed state sequence τ belongs to $\mathcal{T}$ iff $\mu(\tau(0)) \in \mathcal{I}$ and there exists an interval sequence $\overline{I} = I_0 I_1 I_2 \ldots$ of right-closed (or unbounded) intervals that is compatible with τ and satisfies the following conditions:

  1. (*observation consecution*) For all $t \in I_i$ and $t' \in I_{i+1}$, there is a transition $e \in \mathcal{E}$ such that $\mu(\tau(t')) \in e(\mu(\tau(t)))$. The transition e is *taken* at time $r(I_i)$.
  2. (*delay consecution*) For all $t \in I_i$, if the transition $e \in \mathcal{E}$ is enabled on $\mu(\tau(t))$ and not taken at time $l(I_i)$, then $\delta_e(\tau(t)) = \delta_e(\tau(l(I_i))) + t - l(I_i)$; if e is enabled on $\mu(\tau(t))$ and taken at time $l(I_i)$, then $\delta_e(\tau(t)) = t - l(I_i)$; otherwise, $\delta_e(\tau(t)) = 0$.

3. (*timing*) If the transition $e \in \mathcal{E}$ is taken at time $t \in \mathrm{R}$, then $l_e \leq \delta_e(\tau(t)) \leq u_e$. In other words, once enabled, e is delayed for at least l_e time units and at most u_e time units (if $u_e \in \mathrm{N}$).
4. (*fairness*) If the transition $e \in \mathcal{E}$ with $u_e = \infty$ is not taken after time $t \in \mathrm{R}$, then e is not enabled on $\mu(\tau(t'))$ for some later time $t' \geq t$.
5. (*termination*) If $\overline{I} = I_0 I_1 \ldots I_n$ is finite and $t \in I_n$, then no transition $e \in \mathcal{E}$ is enabled on $\mu(\tau(t))$.

It is not difficult to check that the set $\mathcal{T}$ of timed state sequences is fusion-closed.

Translation to timed automata

If the set $\mathcal{P}$ of propositions is finite, then the real-time system $\mathrm{S_T}$ can, alternatively, be defined by a timed automaton $\mathrm{A_T}$. For simplicity, let us assume that all transitions of T become disabled once they are taken; that is, $(\sigma, \sigma') \in e$ implies $e(\sigma') = \emptyset$. The timed automaton $\mathrm{A_T} = (\mathcal{P}, Q, Q_0, C, \mu, \nu, E, F)$ contains a location for each observation-transition pair and a clock for each transition of the timed transition system T:

$Q = 2^{\mathcal{P}} \times \mathcal{E}$;
$Q_0 = \mathcal{I} \times \mathcal{E}$;
$C = \{x_e \mid e \in \mathcal{E}\}$;
$\mu(\sigma, e) = \bigwedge_{p \in \sigma} p \wedge \bigwedge_{p \notin \sigma} \neg p$ for all locations $(\sigma, e) \in Q$.

A location (σ, e) of the timed automaton $\mathrm{A_T}$ records the current observation σ and the transition e that was taken most recently. The automaton $\mathrm{A_T}$ contains an edge from the location (σ, e) to the location (σ', e') iff $\sigma' \in e'(\sigma)$; that is, if the transition e' leads from the observation σ to the observation σ'. Along this edge, all clocks $x_{e''}$ whose corresponding transitions e'' are disabled on σ but enabled on σ' are reset. Thus, for any currently enabled transition e, the clock x_e shows the time that has elapsed since e became enabled. Now it is not hard to enforce all minimal and maximal delays:

- Each location $(\sigma, e) \in Q$ is labeled with the timing constraint $x_e > l_e$; this ensures the lower-bound requirements.
- In addition, each location $(\sigma, e) \in Q$ is labeled with the timing constraint $x_{e'} \leq u_{e'}$ for every transition e' that is enabled on σ (if $u_{e'} \in \mathrm{N}$); this enforces the finite upper-bound requirements.
- Finally, the acceptance family F of the timed automaton $\mathrm{A_T}$ contains all subsets $F_i \subseteq Q$ such that for every transition $e \in \mathcal{E}$ with an infinite maximal delay, there is a location $(\sigma, e') \in F_i$ and either e is not enabled on σ or $e = e'$. This construction takes care of the infinite upper-bound requirement that the transition e is either disabled infinitely often or taken infinitely often.

Interleaving semantics

We presented an interval-based strictly-monotonic real-time semantics for timed transition systems. The alternatives include, as usual, interleaving (i.e., weakly-monotonic), fictitious-clock, and synchronous semantics. Indeed, timed transition systems are generally given an interleaving semantics, which allows finitely many transitions to be taken at the same time.

The advantage of an interleaving semantics for transition systems becomes apparent when we attempt to compose two systems in parallel. In the untimed case, the size of the product system explodes without interleaving, because it must contain a new transition for every pair of transitions that may be taken simultaneously. In the timed case, the product cannot even be defined in this manner, because a single minimal delay and a single maximal delay would have to be assigned to the joint transitions. By employing an interleaving semantics, on the other hand, the transition set of the product system is simply the union of the transition sets for the component systems. If desired, one may define additional joint transitions for a few pairs of transitions that must be taken simultaneously, such as synchronization transitions. For concrete examples of how timed transition systems (with an interleaving semantics) can be used to model real-time systems, we refer to the article *Timed Transition Systems* in this volume.

5 Verification Results

We presented several automata-based and logic-based languages for the description of real-time systems and timing requirements. We now survey the results concerning the verification of systems that are defined in these formalisms. The verification problem assumes two descriptions of real-time behavior, I and S, and asks if I conforms with (satisfies, refines, implements) S. Typically, the implementation I describes a real-time system and the specification S describes timing requirements of I. We distinguish between homogeneous verification methods, which assume that both the implementation and the specification are given in the same language, and heterogeneous methods, for the case that that the system and its requirements are defined in different formalisms.

5.1 Homogeneous verification 1: Logics

If both the implementation and the specification are given as formulas ϕ_I and ϕ_S, respectively, of a logic $\mathcal{L}$, then the system I meets the specification S iff the implication

$$\phi_I \rightarrow \phi_S$$

is valid; or, equivalently, iff the conjunction

$$\phi_I \wedge \neg\phi_S$$

is unsatisfiable. Consequently, we may use (1) decision procedures for $\mathcal{L}$ to solve the verification problem algorithmically, and (2) proof systems for $\mathcal{L}$ to solve the verification problem deductively.

Decidability of real-time logics

Only propositional versions of temporal logics are decidable (cf. [Aba87]). With regard to real-time extensions of propositional temporal logics, there are two parameters that determine the decidability of a language — the domain *Dom* of time and the operations *Ops* on time. Let $\mathcal{L}_{Dom,Ops}$ be an extension of a propositional linear-time or branching-time logic employing any of the three notations for writing timing constraints:

- The time domain *Dom* is a semantic parameter that defines the models of $\mathcal{L}_{Dom,Ops}$. We distinguish between dense time domains $Dom = Dense$, such as the nonnegative reals, and discrete time domains $Dom = Discrete$, such as the nonnegative integers.

- The set *Ops* of operations on time is a syntactic parameter that defines the formulas of $\mathcal{L}_{Dom,Ops}$; it contains the primitive predicates and functions that occur in the atomic timing constraints of $\mathcal{L}_{Dom,Ops}$. The order predicate $\leq$ and addition $+c$ by integer constants are necessary to specify constant lower and upper bounds on the time distance between events. Thus, the minimal set of real-time operators we consider is the set $Ops = Succ$, which contains these primitives. Addition by constants is a binary notation for iterated successor operations; we will use the standard assumption of binary encoding of time constants for classifying the complexity of problems and algorithms. Bounded-operator logics fall into the class $\mathcal{L}_{Dom,Succ}$, because every bounded-operator formula (with constant time bounds on the temporal operators) can be rewritten with freeze quantifiers or a clock variable and timing constraints over the operations from *Succ*.

  More complicated timing constraints can be expressed using real-time operators from the set $Plus = \{\leq, +, 0, 1\}$. Addition on time is needed, for example, to specify the property that the distance between successive events remains constant.

There is an intrinsic characterization of the timing requirements whose satisfiability can be decided. This characterization is independent of the details of any particular real-time logic as long as the logic is sufficiently expressive to define punctuality properties of the form

$$\Box(p \rightarrow \Diamond_{=n}\, q),$$

which requires that every event p is followed by an event q after *precisely* n time units, for some constant n. Punctuality is definable in all logics we presented, with the exception of MITL.

Let $\mathcal{L}$ be a language that (1) is closed under all boolean operations and (2) can express punctuality. Then the satisfiability problem is undecidable for both $\mathcal{L}_{Dense,Succ}$ and $\mathcal{L}_{Discrete,Plus}$. In fact, if infinite recurrence (i.e., $\Box\Diamond p$) is definable in $\mathcal{L}$, then the complexity of the satisfiability problem is located deep in the hierarchy of undecidable problems — Σ_1^1-complete, to be precise (cf. [Rog67]). This result was first obtained for linear temporal logic with freeze quantification [AH89] and immediately applies to both explicit-clock and branching-time logics. The undecidability of time-bounded operators over a dense time domain was proved along the same lines [AH90].

Complexity of discrete-time logics

The satisfiability problem is, on the other hand, decidable for $\mathcal{L}_{Discrete,Succ}$, for all logics $\mathcal{L}$ we presented. For linear-time logics, the exact complexity of the satisfiability problems is known and independent of interval-based or point-based, synchronous (i.e., strictly-monotonic) or asynchronous (i.e., weakly-monotonic) interpretation: EXPSPACE-complete for TPTL [AH89] and MTL [AH90]; nonelementary for RTTL [AH90]. These results show the freeze quantification of TPTL to be superior to the classical quantification of RTTL. The extra exponential on top of the untimed linear temporal logic PTL, which can be decided in PSPACE, is inherent to real-time reasoning with binary encoding of time constants. The set *Ops* of primitive operations on time for the logics TPTL, MTL, and RTTL also includes congruence modulo constants ($\Box_{\equiv_0 2}$ means "in all states with an even time difference from the current state"); introducing this primitive does not affect the complexity of the logics.

The papers cited above give doubly-exponential-time decision procedures for both TPTL and MTL and compare both logics. The verification algorithm for MTL is somewhat less expensive than the algorithm for TPTL, as the first one depends exponentially on the value of the largest time constant involved and the latter depends exponentially on the value of the product of all time constants. In addition, while MTL includes past temporal operators, which do not affect its complexity, the addition of past operators renders the satisfiability problem for TPTL nonelementary [AH90]. On the other hand, even though we will see (in Section 6) that the same timing requirements are definable in both TPTL and MTL, we observed that TPTL specifies nonlocal timing constraints more succinctly. Thus, we conclude that, for a given specification and verification task, either one of TPTL and MTL may be more suitable.

For completeness, we include two related results. First, the satisfiability problem for the quantifier-free explicit-clock logic XCTL is PSPACE-complete, despite the admission of addition over time [HLP90]. XCTL, however, is a language that is not closed under negation and, hence, cannot be used to solve the homogeneous verification problem. Second, the satisfiability problem for the branching-time bounded-operator logic RTCTL is doubly-exponential-time-complete [EMSS89]. Little else is known about the precise complexity of branching-time logics for timed reasoning.

Towards dense-time logics

The proof of the undecidability of real-time logics over a dense time domain makes crucial use of punctuality properties. The bounded-operator logic MITL originated in an effort to define a nontrivial real-time logic that cannot express punctuality requirements and, indeed, the satisfiability problem for MITL was recently shown to be EXPSPACE-complete [AFH91]. The doubly-exponential-time verification algorithm for MITL, which is the first such algorithm for a linear-time logic over a dense time domain, is considerably more complex than discrete-time algorithms and it is not yet fully understood precisely which real-time properties (which superset of MITL) can be verified by this method.

Axiomatization of real-time logics

The Σ_1^1-hardness results mentioned above imply that there cannot be complete proof systems for many logics over a dense time domain as well as logics with addition on time. In the case of discrete time, a complete finite axiomatization is known for TPTL [Hen90]. That axiom system characterizes the freeze quantifier as a construct of modal logic independent of the notion of time. For time-bounded operators, several axioms have been suggested without claim of completeness [Koy90]. The authors suspect that MTL has a clean, complete, finite axiomatization, which is yet to be found.

5.2 Homogeneous verification 2: Automata

Assuming a trace semantics, the implementation I defines a set L_I of generated timed observation sequences and the specification S defines a set L_S of admitted timed observation sequences. In this case, the verification problem reduces to the problem of checking the containment

$$L_I \subseteq L_S$$

between sets of timed observation sequences. Linear-time logics, timed automata, and timed transition systems all define sets of timed observation sequences. In the previous subsection, we discussed the case that both L_I and L_S are given by formulas of a linear-time logic. In this subsection, we discuss trace verification for the case that both the implementation and the specification are given by timed automata.

A real-time system I often is defined naturally by a finite set $\{\mathrm{A}_I^1, \ldots \mathrm{A}_I^m\}$ of timed automata, each of which represents a parallel process of I. The generated timed observation sequences correspond, then, to the runs of the product automaton

$$A_I = A_I^1 \times \cdots \times A_I^m.$$

A specification S that is given as another timed automaton, A_S, is violated by any timed observation sequence that corresponds to a run of the complementary automaton $\neg A_S$. Thus, the implementation I meets the specification S iff the product automaton

$$A_I \times \neg A_S$$

has no run. Consequently, the verification problem can be solved by algorithms for (1) constructing the product of timed automata, (2) complementing a timed automaton, and (3) checking if a timed automaton defines the empty language.

Algorithms for constructing the product of timed automata and for checking emptiness of timed automata were given for a point-based strictly-monotonic real-time interpretation [AD90] and can easily be modified for other semantical assumptions such as time intervals, interleaving, or discrete time:

- As in the untimed case, the number of locations of the product automaton is proportional to the product of the number of locations for the component automata. Hence, the size of the implementation automaton is exponential in the description of the individual processes. This blow-up is known as the state explosion problem.
- The problem of checking if a timed automaton A defines the empty language is PSPACE-complete, provided that the timing constraints of A contain only operations from *Succ*. Assuming binary encoding of time constants, the running time of the algorithm for checking the emptiness of A depends exponentially on the length of the timing constraints for A. Similar to the satisfiability problem for real-time logics, checking emptiness is undecidable for timed automata with addition of clock variables, even in the case of discrete time [Alu91].

This leaves the task of complementing timed automata. Unfortunately, over a dense time domain, timed (Muller) automata are not closed under complement [AD90]. There are three options for circumventing this problem:

1. We may choose the fictitious-clock abstraction of a discrete time domain, which allows the determinization and complementation of timed (Muller) automata just as in the untimed case.
2. We may restrict ourselves to specifications that are given by deterministic timed automata, which are trivially complementable. In this case, the overall time complexity of checking the containment $L_I \subseteq L_S$ is exponential in both the descriptions of L_I and L_S.
3. We may define the set L_S of admitted timed observation sequences by the temporal logic MITL. This option will be discussed in the following subsection.

In each of the three cases, the implementation language L_I can, alternatively, be defined by a timed transition system.

5.3 Heterogeneous verification

Finally, we discuss the case that the implementation is given by a product of timed automata or by a timed transition system, and the specification is given by a logical formula. The corresponding verification problem can be solved either algorithmically, by so-called model checking procedures, or deductively.

Model checking

In the case of linear-time logics, the model checking problem is equally difficult as checking satisfiability; that is, it is undecidable for dense-time logics capable of expressing punctuality and EXPSPACE-complete for the discrete-time logics TPTL [AH89] and MTL [AH90] and the dense-time logic MITL [AFH91]. Given a specification S by a formula ϕ_S, the decision procedures for TPTL, MTL, and MITL construct from the negated formula $\neg\phi_S$ a timed automaton $\mathrm{A}_{\neg\phi_S}$ whose runs correspond precisely to the timed observation sequences that violate the specification S. We can then proceed as in the homogeneous case and check if the product automaton

$$\mathrm{A}_I \times \mathrm{A}_{\neg\phi_S}$$

has a run. It follows that the running time of the model checking algorithms depends singly exponentially on the size of the implementation and doubly exponentially on the size of the specification formula. For the logic XCTL, a different procedure allows the model checking of timing properties that contain addition, in PSPACE [HLP90].

Perhaps surprisingly, in the case of branching-time logics, model checking of punctuality properties is possible even over a dense time domain. The model checking problem is PSPACE-complete for the branching-time logic TCTL of the type $\mathcal{L}_{Dense,Succ}$ for a bounded-operator as well as a half-order syntax [ACD90, Alu91]. The running time of the verification algorithm depends exponentially on the length of the timing constraints in both the implementation and the specification. The complexity of branching-time model checking remains the same under a synchronous discrete-time interpretation, as was observed earlier [EMSS89]. Recently, a model-checking algorithm has been designed for expressions of a dense-time process algebra over timed automata [NSY].

Temporal proof rules

Two important classes of timing requirements are *bounded-response* properties, such as

$$\Box(p \rightarrow \Diamond_{\leq 3}\, q), \tag{$\dagger$}$$

and *bounded-invariance* properties, such as

$$\Box(p \rightarrow \Box_{<3}\, \neg q).$$

Bounded-response properties assert that "something good" will happen within a specified amount of time; bounded-invariance properties assert that "nothing bad" will happen for a certain amount of time. In other words, bounded-response and bounded-invariance properties define upper and lower bounds, respectively, on the time distance between events (such as the events p and q of our sample formulas).

Bounded-response and bounded-invariance properties of timed transition systems can be proved in two different ways. First, it is a well-known observation that both classes of properties are safety properties under the assumption that time progresses [LA90, Hen91a, Lam91]. For example, the bounded-response property ($\dagger$) that was expressed by a "liveness-like" bounded-operator formula (employing a time-bounded version of the liveness operator $\Diamond$) can alternatively be specified by an explicit-clock formula that uses the safety operator U (*unless*; cf. [MP83]):

$$\forall x.\, \Box((p \wedge T = x) \rightarrow (T < x + 3)\, \mathrm{U}\, q).$$

This formula asserts that if a stimulus p happens at time x, then from this point on the time will not reach $x + 3$ either forever (which is impossible because time must progress) or until the response q

happens. Consequently, q must occur within at most 3 time units from p. This translation shows that no new proof rules are necessary for the *explicit-clock* style of timed verification, which proves safety formulas containing the clock variable T by assertional reasoning [Haa81, SL87, HMP91, Hoo91]. All time-bounded properties of timed transition systems can, in principle, be verified using a standard, uniform set of untimed temporal-logic rules (e.g., [MP89]).

In the bounded-operator notation, on the other hand, upper-bound properties bear a close resemblance to liveness properties and lower-bound properties closely resemble safety properties. This similarity may be cultivated using separate bounded-operator proof principles for the classes of bounded-response and bounded-invariance properties: the standard temporal-logic rules for the untimed response and invariance classes can be decorated with time bounds on the temporal operators [HMP91]. Relative completeness (with respect to state reasoning, which need not be propositional) of this *bounded-operator* style of timed verification was shown for a restricted class of timed transition systems [Hen91b]. Deductive bounded-operator reasoning has not been explored yet for more complicated real-time properties than bounded response and bounded invariance.

6 Expressiveness Results

We now compare the expressive power of the real-time specification languages presented in this paper. The questions regarding the expressiveness of linear-time formalisms versus branching-time formalisms have been studied in the untimed case (cf. [Eme90]). As the introduction of real-time considerations does not seem to raise any new questions in this context, we restrict our attention to the linear-time case.

Given a semantics SEM, an expression ϕ of a linear-time specification language LAN defines a set L_ϕ of timed observation sequences (or weakly-monotonic timed observation sequences, in the interleaving case). The set L_ϕ is called a *real-time property* or, more specifically, a SEM-property. The expressive power of the language LAN under the semantic assumption SEM is measured as the set

$$\mathbf{LAN}_{\mathrm{SEM}} = \{L_\phi \mid \phi \in \mathrm{LAN}\}$$

of SEM-properties that are definable in LAN.

As syntactic options, we consider the languages of the logics MTL, TPTL, RTTL, XCTL, and MITL, as well as timed automata (TA) and deterministic timed automata (DTA) whose assertion language for timing constraints is that of TPTL (and RTTL). Since we are interested primarily in questions concerning time, we concentrate on the fictitious-clock abstraction. Accordingly, we consider the two semantic options of real-numbered time and integer time. A set of timed observation sequences over the time domain R is a *dense-time* property; a set of timed observation sequences over N, a *discrete-time* property. For example, we write $\mathbf{MITL}_{\mathrm{R}}$ for the set of dense-time properties that can be defined by MITL-formulas. Similarly, $\mathbf{DTA}_{\mathrm{N}}$ stands for the set of discrete-time properties that are definable by deterministic timed automata.

We first compare all languages assuming a common semantics. Then we compare the expressive power of similar languages with different semantics, with the goal of characterizing the loss of expressiveness introduced by the semantic abstraction of a fictitious clock.

6.1 Comparison of syntax

We presented various ways of writing timing constraints; they include the bounded-operator, half-order, and explicit-clock notations for temporal logics, and timed automata. All of these notations can be interpreted over integer time and over real-numbered time.

Integer time

To compare the different notations, it is best to study the underlying logical theory of timed observation sequences over the time domain N. The theory of the natural numbers with linear order and monadic predicates underlies linear temporal logic (cf. [GPSS80]). We combine this theory of observation sequences with a theory of integer time, via a monotonic function that maps every observation to its time. The timing constraints are restricted to use only the ordering, successor, and congruence operations on time. The resulting second-order theory $\mathbf{T}^2_{\mathrm{N}}$ (with quantification over the monadic predicates) — the *theory of discrete timed observation sequences* — is decidable; its expressive power can be characterized by ω-regular sets employing auxiliary propositions that record some finite information about the time differences between neighboring observations [AH90]. Using the theory $\mathbf{T}^2_{\mathrm{N}}$ as point of reference, we have the following results.

1. All the three logical notations are equally expressive as the first-order fragment $\mathbf{T}_{\mathrm{N}}$ of the theory $\mathbf{T}^2_{\mathrm{N}}$ of discrete timed observation sequences:

$$\mathbf{T}_{\mathrm{N}} = \mathbf{MTL}_{\mathrm{N}} = \mathbf{TPTL}_{\mathrm{N}} = \mathbf{RTTL}_{\mathrm{N}}.$$

   Since the decision problem for $\mathbf{T}_{\mathrm{N}}$ is nonelementary, both MTL and TPTL (but not RTTL) characterize comparatively tractable and expressively complete fragments of $\mathbf{T}_{\mathrm{N}}$. As in the untimed case, the expressive power of the second-order theory $\mathbf{T}^2_{\mathrm{N}}$ can be attained by introducing quantification over propositions or the grammar operators of the extended temporal logic of [Wol83]. These expressiveness results were obtained under a point-based weakly-monotonic semantics [AH90], but they apply equally to the interval-based and the synchronous case.

2. It is not hard to show that timed automata, too, identify an expressively complete fragment of the second-order theory of discrete timed observation sequences:

$$\mathbf{T}^2_{\mathrm{N}} = \mathbf{DTA}_{\mathrm{N}} = \mathbf{TA}_{\mathrm{N}}.$$

Since the class $\mathbf{T}^2_{\mathrm{N}}$ is so robust, closed under all boolean operations, and emptiness is elementarily decidable for reasonable definition languages, it provides a clean notion of finite-state property for integer time: a discrete-time property L is *finite-state* iff $L \in \mathbf{T}^2_{\mathrm{N}}$. Overall, the untimed theoretical properties of observation sequences generalize conservatively to timed observation sequences over a discrete time domain.

The expressive power of the logic XCTL, the quantifier-free fragment of RTTL with addition over time, is incomparable to the class $\mathbf{T}_{\mathrm{N}}$ [HLP90]. On one hand, XCTL forbids quantification over the time variables; on the other hand, it permits stronger timing constraints that involve the use of addition.

Real-numbered time

Much less is known about the relative expressive power of the various languages if they are interpreted over timed observation sequences with real-numbered time. Since the satisfiability problem is undecidable for MTL over dense time and the class $\mathbf{TA}_{\mathrm{R}}$ is not closed under complementation, the authors looked for less expressive languages:

$$\mathbf{MITL}_{\mathrm{R}} \subset \mathbf{MTL}_{\mathrm{R}} \quad \text{and} \quad \mathbf{DTA}_{\mathrm{R}} \subset \mathbf{TA}_{\mathrm{R}}$$

[AFH91, AD90] (by $\subset$, we denote strict containment). Since also

$$\mathbf{MITL}_{\mathrm{R}} \subset \mathbf{TA}_{\mathrm{R}},$$

both classes $\mathbf{MITL}_{\mathrm{R}}$ and $\mathbf{DTA}_{\mathrm{R}}$ do have all desired closure and decidability properties; unfortunately, neither one is a subset of the other: since MITL prohibits timing constraints that involve equality (i.e., singular intervals), the two classes $\mathbf{MITL}_{\mathrm{R}}$ and $\mathbf{DTA}_{\mathrm{R}}$ are incomparable. Furthermore, both classes seem quite nonrobust, with heavy dependence on syntactic idiosyncrasies. Thus, the main question the authors would like to see answered [AH91] remains open:

> *Is there an agreeable notion of finite-state property for real-numbered time?* The set of such properties ought to be closed under all boolean operations, have a characterization with an elementarily decidable emptiness problem, and be, in a suitable sense, "maximal."

Another interesting question asks how the bounded-operator notation compares with the half-order notation. We know that they are equally expressive in the case of integer time, but the proof makes crucial use of the discreteness of time. Hence, the authors conjecture that freeze quantifiers are more expressive than bounded operators in the case of real-numbered time:

$$\mathbf{MTL}_{\mathrm{R}} \overset{?}{\subset} \mathbf{TPTL}_{\mathrm{R}}.$$

In particular, the nonlocal timing property (‡) of Subsection 3.2 is suspected to be inexpressible by the bounded-operator notation of MTL.

6.2 Comparison of semantics

Motivated by the result that the verification of punctuality properties is undecidable over a dense time domain, we presented two solutions for obtaining decidable dense-time logics. First, we weakened the expressiveness of languages, such as MTL, by adopting the *semantic* abstraction of a fictitious clock. Second, we weakened the expressiveness of MTL by adopting the *syntactic* concession of prohibiting singular intervals in MITL. Both the semantic abstraction of digitizing models as well as the syntactic restriction of excluding equality in timing constraints limit the real-time properties that are definable in a similar way: they rule out the notion of absolute punctuality and replace it by a looser concept of *almost*-on-time behavior. This sacrifice is viable because, by choosing the clock tick of the fictitious clock small enough, we can still achieve arbitrary precision in either approach. Moreover, the corresponding costs for achieving the desired accuracy are the same. This raises the question if one technique is superior to the other in expressive power. In other words, how do the two classes $\mathbf{MTL}_{\mathrm{N}}$ and $\mathbf{MITL}_{\mathrm{R}}$ compare?

To relate the expressiveness of two languages under different semantical assumptions, we have to put them on common ground. A semantical abstraction, such as a fictitious clock, is an equivalence relation on the set of timed observation sequences over the time domain R; it does not discriminate between timed observation sequences within the same equivalence class. For instance, the fictitious clock that ticks every 0.5 time units (beginning at time 0) cannot distinguish between the two timed observation sequences

$$(\{p\}, 0.2) \to (\{q\}, 1) \to (\{p\}, 5.9),$$
$$(\{p\}, 0.4) \to (\{q\}, 1) \to (\{p\}, 5.8).$$

The dense-time property defined by an expression ϕ of a language LAN under a semantic abstraction SEM is, then, the union of the SEM-equivalence classes of the models of ϕ. This approach determines the *absolute* expressive power of the syntax-semantics pair (LAN, SEM) in terms of which dense-time properties are definable.

The authors know of only one result that relates the expressive power of different models. MITL is, in the above sense, strictly more expressive than MTL with respect to a fictitious clock of any stepwidth; that is, the dense-time properties that can be defined in MITL are a proper superset of those definable with equality under a fictitious-clock interpretation [AFH91]. Also, many of the practically interesting forms of punctuality are still expressible in MITL, such as the requirement that every event p is separated from the *closest* subsequent event q by precisely 3 time units. These observations suggest a more thorough study of real-numbered time in the directions taken by MITL and timed automata, rather than a surrender to fictitious-clock models.

References

[Aba87] M. Abadi. *Temporal-Logic Theorem Proving.* PhD thesis, Stanford University, 1987.

[ACD90] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking for real-time systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.

[AD90] R. Alur and D.L. Dill. Automata for modeling real-time systems. In M.S. Paterson, editor, *ICALP 90: Automata, Languages, and Programming*, Lecture Notes in Computer Science 443, pages 322–335. Springer-Verlag, 1990.

[AFH91] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the Tenth Annual Symposium on Principles of Distributed Computing*, pages 139–152. ACM Press, 1991.

[AH89] R. Alur and T.A. Henzinger. A really temporal logic. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 164–169. IEEE Computer Society Press, 1989.

[AH90] R. Alur and T.A. Henzinger. Real-time logics: complexity and expressiveness. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 390–401. IEEE Computer Society Press, 1990.

[AH91] R. Alur and T.A. Henzinger. Time for logic. *SIGACT News*, 22(3):6–12, 1991.

[Alu91] R. Alur. *Techniques for Automatic Verification of Real-time Systems.* PhD thesis, Stanford University, 1991.

[BH81] A. Bernstein and P.K. Harter, Jr. Proving real-time properties of programs with temporal logic. In *Proceedings of the Eighth Annual Symposium on Operating System Principles*, pages 1–11. ACM Press, 1981.

[BMP81] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. In *Proceedings of the Eighth Annual Symposium on Principles of Programming Languages*, pages 164–176. ACM Press, 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[Dil89] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *CAV 89: Automatic Verification Methods for Finite-state Systems*, Lecture Notes in Computer Science 407, pages 197–212. Springer-Verlag, 1989.

[EC82] E.A. Emerson and E.M. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers (North-Holland), 1990.

[EMSS89] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. Presented at the First Annual Workshop on Computer-aided Verification, Grenoble, France, 1989.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the Seventh Annual Symposium on Principles of Programming Languages*, pages 163–173. ACM Press, 1980.

[Haa81] V.H. Haase. Real-time behavior of programs. *IEEE Transactions on Software Engineering*, SE-7(5):494–501, 1981.

[Har88] E. Harel. Temporal analysis of real-time systems. Master's thesis, The Weizmann Institute of Science, Rehovot, Israel, 1988.

[Hen90] T.A. Henzinger. Half-order modal logic: how to prove real-time properties. In *Proceedings of the Ninth Annual Symposium on Principles of Distributed Computing*, pages 281–296. ACM Press, 1990.

[Hen91a] T.A. Henzinger. Sooner is safer than later. Technical report, Stanford University, 1991.

[Hen91b] T.A. Henzinger. *The Temporal Specification and Verification of Real-time Systems.* PhD thesis, Stanford University, 1991.

[HLP90] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit-clock temporal logic. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 402–413. IEEE Computer Society Press, 1990.

[HMP90] T.A. Henzinger, Z. Manna, and A. Pnueli. An interleaving model for real time. In *Proceedings of the Fifth Jerusalem Conference on Information Technology*, pages 717–730. IEEE Computer Society Press, 1990.

[HMP91] T.A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proceedings of the 18th Annual Symposium on Principles of Programming Languages*, pages 353–366. ACM Press, 1991.

[Hoo91] J. Hooman. *Specification and Compositional Verification of Real-time Systems.* PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 1991.

[HW89] J. Hooman and J. Widom. A temporal-logic-based compositional proof system for real-time message passing. In E. Odijk, M. Rem, and J.-C. Syre, editors, *PARLE 89: Parallel Architectures and Languages Europe*, vol. II, Lecture Notes in Computer Science 366, pages 424–441. Springer-Verlag, 1989.

[KdR85] R. Koymans and W.-P. de Roever. Examples of a real-time temporal specification. In B.D. Denvir, W.T. Harwood, M.I. Jackson, and M.J. Wray, editors, *The Analysis of Concurrent Systems*, Lecture Notes in Computer Science 207, pages 231–252. Springer-Verlag, 1985.

[Kel76] R.M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time Systems*, 2(4):255–299, 1990.

[KVdR83] R. Koymans, J. Vytopil, and W.-P. de Roever. Real-time programming and asynchronous message passing. In *Proceedings of the Second Annual Symposium on Principles of Distributed Computing*, pages 187–197. ACM Press, 1983.

[LA] L. Lamport and M. Abadi. Refining and composing real-time specifications. This volume.

[LA90] N.A. Lynch and H. Attiya. Using mappings to prove timing properties. In *Proceedings of the Ninth Annual Symposium on Principles of Distributed Computing*, pages 265–280. ACM Press, 1990.

[Lam83] L. Lamport. What good is temporal logic? In R.E.A. Mason, editor, *Information Processing 83: Proceedings of the Ninth IFIP World Computer Congress*, pages 657–668. Elsevier Science Publishers (North-Holland), 1983.

[Lam91] L. Lamport. The temporal logic of actions. Technical report, DEC Systems Research Center, Palo Alto, California, 1991.

[Lew90] H.R. Lewis. A logic of concrete time intervals. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 380–389. IEEE Computer Society Press, 1990.

[LPZ85] O. Lichtenstein, A. Pnueli, and L.D. Zuck. The glory of the past. In R. Parikh, editor, *Logics of Programs*, Lecture Notes in Computer Science 193, pages 196–218. Springer-Verlag, 1985.

[MP83] Z. Manna and A. Pnueli. Proving precedence properties: the temporal way. In J. Diaz, editor, *ICALP 83: Automata, Languages, and Programming*, Lecture Notes in Computer Science 154, pages 491–512. Springer-Verlag, 1983.

[MP89] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, Lecture Notes in Computer Science 354, pages 201–284. Springer-Verlag, 1989.

[NSY] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. This volume.

[Ost90] J.S. Ostroff. *Temporal Logic of Real-time Systems*. Research Studies Press, 1990.

[PdR82] A. Pnueli and W.-P. de Roever. Rendez-vous with Ada: a proof-theoretical view. In *Proceedings of the SIGPLAN AdaTEC Conference on Ada*, pages 129–137. ACM Press, 1982.

[PH88] A. Pnueli and E. Harel. Applications of temporal logic to the specification of real-time systems. In M. Joseph, editor, *Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science 331, pages 84–98. Springer-Verlag, 1988.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[Pnu86] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Current Trends in Concurrency*, Lecture Notes in Computer Science 224, pages 510–584. Springer-Verlag, 1986.

[Rog67] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill Book Company, 1967.

[Ron84] D. Ron. Temporal verification of communication protocols. Master's thesis, The Weizmann Institute of Science, Rehovot, Israel, 1984.

[SL87] A.U. Shankar and S. Lam. Time-dependent distributed systems: proving safety, liveness, and timing properties. *Distributed Computing*, 2(2):61–79, 1987.

[SPE84] D.E. Shasha, A. Pnueli, and W. Ewald. Temporal verification of carrier-sense local area network protocols. In *Proceedings of the 11th Annual Symposium on Principles of Programming Languages*, pages 54–65. ACM Press, 1984.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science Publishers (North-Holland), 1990.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983.