Time for Logic[*]

Rajeev Alur
alur@cs.stanford.edu

Thomas A. Henzinger
tah@cs.stanford.edu

Stanford University

This column is about the problem of verifying that computing systems relate properly to the passage of physical time. While the crucial importance of the real-time verification problem has long been realized in practice, the problem has stirred the interest of the theory community only recently, and only halfheartedly at that. As far as the authors can tell, the two main arguments that are usually brought forward to justify this negligence run as follows:

1. *There is no real-time verification problem.* The real-time behavior of a program depends on myriad factors such as the operating system and the underlying hardware. Worse, many of these factors are usually hidden from the programmer. If your program misses a real-time deadline by a constant factor, use (or wait for) a faster machine.

While this analysis of the problem is correct, the conclusion is not. Indeed, the real-time verification problem is far messier than the convenient abstraction of time by traditional program verification techniques. This observation only goes to show how clever these techniques are — and that they ought to be used whenever possible —, but it does not make the need for real-time systems disappear. Tampering with the speed of the machine is a naive nonsolution for the most important real-time systems, which are embedded in real-time environments whose speed is not under our control.

2. Granted, the verification of timing constraints is a real concern, but *there is nothing special about the real-time verification problem per se.* Time is just another parameter of the system state — treat it as such. (That is, use the standard, established program verification techniques.)

There is an obvious reply: no. Time *is* fundamentally different from the state components of a discrete computing system. For all we know, time is continuous, monotonic, and divergent, and program variables generally happen not to have either one of these characteristics. Only if we recognize the special status of time, will we be able to find and exploit the intricacies of proving timing properties and avoid pitfalls like Zeno behaviors.

There is also a technical argument that substantiates the gut reply. Even if it suffices to model time by a discrete (integer) variable that represents, at all times, the value of some digital clock, the time variable will range over an infinite domain and, thus, real-time

[*]This paper appeared in the logic column, *SIGACT News* **22** (3), 1991, pp. 6–12.

verification will escape all finite-state verification techniques. The authors have shown that this need not be the case. Indeed, with some effort we can make finite-state model-checking techniques work if a program variable ranges over the integers [AH89] or the real numbers [AFH91], provided at most one variable does so and provided the value of the variable is never decreased. Luckily, time happens to be monotonic.

*

Once we have agreed that real-time verification is a genuine object for theoretical investigation, we are immediately confronted with several decisions:

1. *How should we model time?* While physicists seem fairly unanimous in their opinion that, above quantum level, time is best approximated by the real line, philosophers and logicians have proposed a curious variety of different models of time. They range from linear to branching to partial-order time and from discrete to dense to continuous time, to name just a few. We believe that for computer scientists, the choice should be directed by two issues:

   (a) Given a certain mathematical model of physical time, *what can we prove in the model and how difficult is it to do so?* For example, the verification problem of some real-time properties that are undecidable in a dense model of time, can be solved in a discrete model of time [AH89].

   (b) If we have proved something in a certain mathematical model of physical time, *what, if anything, have we proved about physical time?* Needless to point out, "verification" of a real-time system respect to, say, a discrete model of time is only sensible if the result gives us some insight about the physical reality. After all, that flight control system we just proved "correct" has to operate in dense time.

   Of course, to answer these questions, we have to study various models of time and computation, and how they interact. The first question can, for any given model, be formulated and answered in precise, complexity-theoretic terms. The second question must, on the other hand, inevitably remain somewhat vague and philosophical (what is the "physical reality"?) and thus — unfortunately — is often neglected. We believe that the classical physicists' model of time as the real line captures all aspects of reality that we are interested in, and we suggest that it serves as the point of reference for identifying the adequacy of any real-time verification method that deems it convenient to assume, for computational or other reasons, a more abstract representation of time.

2. *How should we specify real-time properties?* The authors have used both temporal logics and finite automata. One commonly raised objection to temporal logic as a real-time specification language is that the very purpose of the temporal operators is the abstraction of time. Since, for the definition of real-time properties, it is necessary to reintroduce time as a first-order domain, one should dispose with the temporal operators altogether. It turns out that this argument leads to unnecessarily unwieldy and expensive specification languages. We have shown that temporal logic allows a more careful, and often more natural, introduction of time than by first-order time

variables, which gives us the additional, crucial benefit of much simpler decision procedures [AH89].

$*$

In the untimed case, one of the most practically successful as well as theoretically appealing verification techniques is the algorithmic verification of *finite-state* systems by model checking. It is, therefore, a prime objective of research in real-time verification to identify the timing properties of systems that can be verified by finite-state techniques. We take this opportunity to present what we (currently) believe is the "right" model of time for finite-state verification, survey what little is known about it, and pose some interesting open problems.

To avoid distraction from the issues that concern time, we choose a simple model of computation: a system is represented by a set of possible behaviors; the behavior of a system is modeled as an ω-sequence of states paired with intervals of the real line. To be precise, a *timed state sequence*

$$(s_0, I_0) \longrightarrow (s_1, I_1) \longrightarrow (s_2, I_2) \longrightarrow (s_3, I_3) \longrightarrow \cdots$$

consists of a sequence $s_0 s_1 \ldots$ of states and a corresponding sequence $I_0 I_1 \ldots$ of time intervals. Each state s_i provides an interpretation for a set of atomic propositions and stays unchanged throughout the time interval I_i, whose end-points are real numbers. Consecutive intervals I_i and I_{i+1} in the sequence are adjacent, and together, all intervals form a partition of the real line. Thus a timed state sequence provides an interpretation for the atomic propositions at every instant of "real" time. Notice that our definition allows state changes to occur at any real point in time, but there can be only finitely many state changes between any two points in time. This condition of *finite variability* makes perfect sense for discrete systems. We make (here) no attempt to model continuous systems, which cannot be described completely by an ω-sequence of state changes.

Real-time specification languages are interpreted over timed state sequences. As an example of a real-time temporal logic, we briefly outline the logic MITL — *metric interval temporal logic* [AFH91]. A standard way of adding timing requirements to temporal languages is to replace the temporal operators by time-constrained versions. For instance, the constrained *eventually* operator $\Diamond_{[2,4]}$ is used to capture the notion "eventually within 2 to 4 time units." The syntax of MITL is defined by extending propositional linear temporal logic by allowing the temporal operators to be constrained by any nonsingular time interval I with integer boundaries. Every formula of MITL defines a set of timed state sequences. For example, the formula $\Diamond_I \phi$ is true at time t of a timed state sequence iff the subformula ϕ is satisfied by the sequence at some later time t' in the interval $t + I$. This interpretation of MITL has the desirable property that the truth of a formula is invariant under *stuttering* (i.e., repetition of states through splitting of the corresponding time intervals).

To see how a typical real-time requirement of a system can be defined in MITL, consider the formula

$$\Box(p \;\rightarrow\; \Diamond_{[2,4]}\, q), \qquad (\dagger)$$

which specifies the bounded-response property that "every p-state is followed by a q-state within 2 to 4 time units."

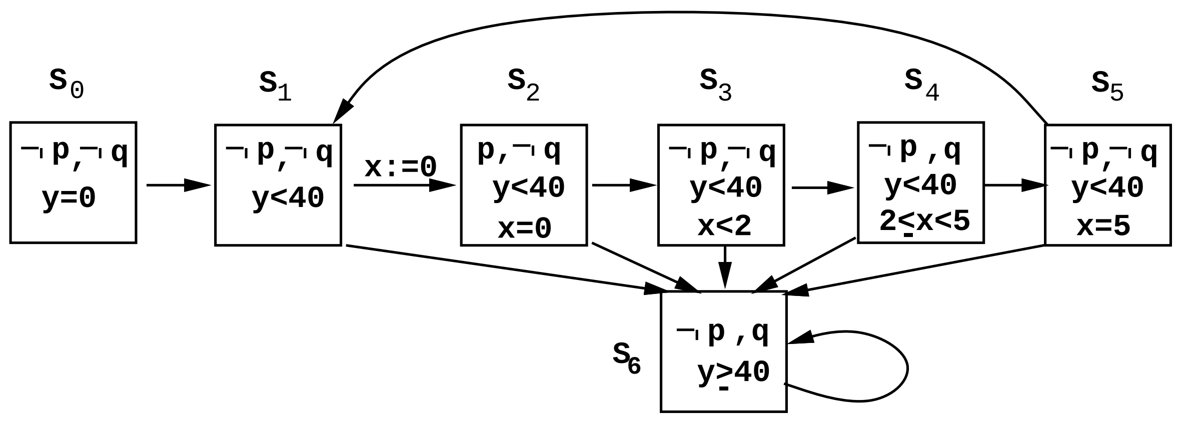

Figure 1: Timed automaton

In [AFH91] we have presented a decision procedure for MITL, which determines if a given formula is satisfied by any timed state sequence; this problem is shown to be EXPSPACE-complete. The blow-up in complexity from the PSPACE-completeness of ordinary propositional temporal logic is due to the logarithmic encoding of the integer constants in time bounds. The same blow-up is observed for logics with a discrete semantics of time [AH90], and seems inherent to all formalisms for real-time reasoning. The decision procedure for MITL crucially depends on the fact that the time intervals that constrain the temporal operators are nonsingular. It turns out that if we allow the operator $\Diamond_{[1,1]}$, which captures equality on time, the decision problem becomes suddenly so wildly undecidable (Σ_1^1-complete) that the resulting logic cannot even be recursively axiomatized.

*

Timed automata [AD90] provide another formalism for defining properties of timed state sequences and are well-suited to model finite-state real-time systems. A timed automaton is a generalization of a nondeterministic finite-state machine over infinite strings. While ω-automata generate (or accept) infinite sequences of states, timed automata are additionally constrained by timing requirements and produce *timed* state sequences. A timed automaton operates with finite control — a finite set of states 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 transition of the automaton may reset some of the clocks; each state 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 state only if the values of the propositions and clocks satisfy the corresponding constraints.

Consider, for example, Figure 1, which shows an automaton M with seven states and two clocks, x and y. The automaton M starts in the initial state s_0 with the clock y initialized to 0. At time 40 the automaton moves to state s_6, and simply loops there. The proposition p denotes an external event that is true only at instantaneous points $t < 40$ in time (and no more than once every 5 time units), namely, whenever M is in state s_2. The automaton responds to p by resetting the clock x, and then it requires that the proposition q holds over the interval $t + [2, 5)$. Thus the automaton M models a system that responds, until time 40, to the event p by setting q to true for the interval $[2, 5)$ following p. In particular, all timed state sequences that are generated by M satisfy the bounded-response property (†).

Timed automata are effectively closed under parallel composition (i.e., product) and projections. The complexity of the emptiness problem, which asks if a given timed automaton generates any timed state sequence, clearly depends on the kind of clock constraints that are admitted. If the only operations on clock values are comparison and addition of constants (i.e., successor), then the emptiness problem for timed automata is PSPACE-complete [AD90]. If we were to allow the addition of clock values, the problem would immediately become highly undecidable (Σ^1_1-complete again). The same phenomenon is observed for real-time formalisms with a discrete semantics of time [AH90]; it seems to be intrinsically impossible to combine reasoning about states with reasoning about addition on time.

*

In our methodology, the real-time verification problem reduces to the following question:

> Given a system M as a product of timed automata and a specification ϕ as a formula of MITL, does every timed state sequence generated by M satisfy ϕ?

This problem is solved in [AFH91] by a model-checking algorithm. The complexity of the algorithm is linear in the number of states of the system M, exponential in the size of the timing constraints of M, and doubly exponential in the length of the specification ϕ. The model-checking problem is also solvable for *specifications* that are given as deterministic timed automata. The *system* may, alternatively, be described as a timed transition system [HMP91] or as an expression of real-time process algebra, which can be compiled into a timed automaton [NSY].

From a theoretical perspective, the picture regarding temporal logics and timed automata is far from complete. In the untimed case, a *property* (or language) is a set of infinite state sequences. There the class of ω*-regular* languages offers a clean notion of "finite-state property." This class is exactly the class of properties that can be defined by ω-regular expressions or, alternatively, by Büchi automata or, alternatively, by formulas of the linear temporal logic ETL or, alternatively, by sentences of the monadic second-order theory S1S of one successor. Moreover, this class is closed under all boolean operations and all problems of relevance to verification are elementarily decidable (for a survey of these and related results, see, for instance, [Em90] and [Th90]).

Let a *real-time property* be a set of *timed* state sequences. The main question we would like to see answered asks

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

The class **TA** of real-time properties that can be defined by timed automata (assuming any standard acceptance condition such as Büchi or Muller acceptance) is unsuitable, because, though closed under union and intersection, it is not closed under complementation; also, important questions such as testing for language universality and language inclusion are undecidable for timed automata. The following two proper subclasses of **TA** look like candidates for the class of real-time properties we are looking for:

1. *Deterministic* timed automata (DTAs) [AD90] are strictly less expressive than (nondeterministic) timed automata. The class **DTA** of DTA-recognizable properties is effectively closed under all boolean operations (but not under projections). It follows that the language universality and inclusion problems are decidable for DTAs.

2. Every property that is expressible in MITL can be defined by a timed automaton, which can be effectively constructed from the given formula [AFH91]. Moreover, the class **MITL** of MITL-definable properties is trivially closed under all boolean operations.

However, since MITL prohibits timing constraints that involve equality (i.e., singular intervals), the two classes **DTA** and **MITL** are incomparable (neither one is a subset of the other). Thus the search for a class of real-time properties that subsumes both of these classes and has all the desired closure and decidability properties must continue.

*

Besides lacking equality, MITL may have a second deficiency as far as its expressive power is concerned. The time-constrained temporal operators of MITL can only relate the times of adjacent temporal contexts. A timed automaton, on the other hand, can start a clock with a transition and check its value, later, at an arbitrarily distant state. In [AH89], we have proposed a way to introduce clocks into temporal logic. The resulting logic is called *timed propositional temporal logic* (TPTL). For example, the bounded-reponse property (†) can be written in (a variant of) TPTL by the formula

$$\Box\, x := 0.\,(p\ \rightarrow\ \Diamond\, y := 0.\,(q \wedge y + 2 \leq x \leq y + 4)),$$

which uses two clocks, x and y. Since it is difficult to syntactically rule out equality in timing constraints, TPTL is undecidable (it was originally proposed as a language to reason about a discrete time domain, where it is decidable). While every formula of MITL can be easily translated into TPTL, we do not know if the expressive power of TPTL, measured as the set of TPTL-definable real-time properties, is strictly greater than that of MITL with equality. Possibly there is a nice, decidable syntactic fragment of TPTL whose definable sets satisfy our demands on the notion of "finite-state real-time property."

References

[AD90] R. Alur, D.L. Dill, "Automata for modeling real-time systems," 17th ICALP, *Lecture Notes in Computer Science* **443**, Springer-Verlag, 1990.

[AH89] R. Alur, T.A. Henzinger, "A really temporal logic," *Proc. of the 30th Annual IEEE Symp. on Foundations of Computer Science*, 1989.

[AH90] R. Alur, T.A. Henzinger, "Real-time logics: Complexity and expressiveness," in *Proc. of the 5th Annual IEEE Symp. on Logic in Computer Science*, 1990.

[AFH91] R. Alur, T. Feder, T.A. Henzinger, "The benefits of relaxing punctuality," *Proc. of the 10th Annual ACM Symp. on Principles of Distributed Computing*, 1991.

[Em90] E.A. Emerson, "Temporal and modal logic," in the *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), Volume B, North-Holland, 1990.

[HMP91] T.A. Henzinger, Z. Manna, A. Pnueli, "Temporal proof methodologies for real-time systems," *Proc. of the 18th Annual ACM Symp. on Principles of Programming Languages*, 1991.

[NSY] X. Nicollin, J. Sifakis, S. Yovine, "From ATP to timed graphs and hybrid systems," in preparation.

[Th90] W. Thomas, "Automata on infinite objects," in the *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), Volume B, North-Holland, 1990.