TAME: A Specialized Specification and Verification System for Timed Automata*

Myla Archer and Constance Heitmeyer
Code 5546, Naval Research Laboratory, Washington, DC 20375
{archer, heitmeyer}@itd.nrl.navy.mil

Abstract

Assuring the correctness of specifications of real-time systems can involve significant human effort. The use of a mechanical theorem prover to encode such specifications and to verify their properties could significantly reduce this effort. A barrier to routinely encoding and mechanically verifying specifications has been the need first to master the specification language and logic of a general theorem proving system. Our approach to overcoming this barrier is to provide mechanical support for producing specifications and verifying proofs, specialized for particular mathematical models and proof techniques. We are currently developing a mechanical verification system called TAME (Timed Automata Modeling Environment) that provides this specialized support using SRI's Prototype Verification System (PVS). Our system is intended to permit steps in reasoning similar to those in hand proofs that use model-specific techniques. TAME has recently been used to detect errors in a realistic example.

1 Introduction

Researchers have proposed many innovative formal methods for developing real-time systems [6]. Such methods can give system developers and customers greater confidence that real-time systems satisfy their requirements, especially their critical requirements. However, applying formal methods to practical systems poses several challenging problems, e.g., how to make formal descriptions and formal proofs understandable to developers and how to design software tools in support of formal methods that are usable by developers.

To address these challenges, we are developing a mechanized system called TAME (Timed Automata Modeling Environment) for specifying and reasoning about real-time systems. Our approach is to build a family of formal reasoning tools customized for specifying and verifying systems represented in terms of specific mathematical models. In [1], we describe how we have built upon the mechanical proof system PVS [14, 15] to support formal specification and verification of real-time systems modeled as Lynch-Vaandrager (LV) timed automata [12, 11]. In [2], we describe how application of the methods of [1], with some minor extensions, was successfully used to detect errors in the specification and correctness proofs of a hybrid system modeled as an LV timed automaton. The tools and methods used in [1] and [2] are part of TAME.

TAME provides mechanical assistance that allows humans to specify and reason about real-time systems in a direct manner. The use of TAME for specification and verification is usually quite straightforward, because TAME provides both definitions for the standard parts together with a structure—in the form of a template to be filled in by the user—for specifying the details of a given model and a set of proof steps suitable for reasoning in that model. By focusing on a particular mathematical model, TAME allows a user to reason within a specialized mathematical framework and avoid having to master the base logic and the complete user interface of the underlying proof system, PVS. By furnishing proof steps designed for checking human-style proofs, TAME facilitates highly useful, conceptual level feedback when errors are discovered.

The next section provides more detail concerning how TAME is built upon PVS. Section 3 describes our experience with the use of TAME, and reports on its current status. Section 4 describes our future plans for TAME. Finally, the relationship of TAME to other efforts is discussed in Section 5.

2 Building on PVS

2.1 PVS

PVS (Prototype Verification System) [15] is a specification and verification environment developed by SRI International's Computer Science Laboratory. In distinction to other widely used proof systems, such as HOL [4] and the Boyer-Moore theorem prover [3], PVS supports *both* a highly expressive specification language and an interactive theorem prover in which most low-level proof steps are automated. The system consists of a specification language, a parser, a type checker, and an interactive proof checker. The PVS specification language is based on a richly typed higher-order logic that permits a type checker to catch a number

*This work is funded by the Office of Naval Research.

of semantic errors in specifications. The PVS prover consists of a set of inference steps that can be used to reduce a proof goal to simpler subgoals that can be discharged automatically by the primitive proof steps of the prover. The primitive proof steps incorporate arithmetic and equality decision procedures, automatic rewriting, and BDD-based boolean simplification.

Specifications in PVS consist of one or more *theories*. Each theory may be parameterized and may import other theories. In proving theorems in PVS, users can apply a sequence of primitive proof steps. In addition to primitive proof steps, PVS supports more complex proof steps called *strategies*, which can be invoked just like any other proof step. Strategies may be defined using primitive proof steps, applicative Lisp code, and other strategies, and may be built-in or user-defined.

2.2 The Relationship of TAME to PVS

To support the mechanization of specifications in a particular mathematical model, TAME provides a template specification corresponding to that model, together with a library of standard theories imported by the template that represent the parts of the mathematical model that all specific instances have in common. The associated specialized proof support is provided by means of user-defined strategies that rely heavily upon the standard theories and the template structure.

3 Current Status of TAME

3.1 Real-Time System Models Supported

Although we eventually plan to extend TAME to other mathematical models, TAME currently provides support for only one: the LV timed automata model. We give some details of this model here, in order to illustrate in Section 3.2 how TAME supports its use, and by analogy, the use of any specific model.

An LV timed automaton is a very general automaton, i.e., a labeled transition system. It need not be finite-state: for example, the state can contain real-valued information, such as the current time and physical quantities that may be involved in a system, such as water levels, railroad gate positions, vehicle accelerations, and so on. The version of this model supported in TAME can be described as follows:

A *timed automaton* A consists of five components:

- $states(A)$, a (finite or infinite) set of states.
- $start(A) \subseteq states(A)$, a nonempty (finite or infinite) set of start states.
- A mapping now from $states(A)$ to $R^{\geq 0}$, the nonnegative real numbers.
- $acts(A)$, a set of actions (or events), which include special *time-passage* actions $\nu(\Delta t)$, where Δt is a positive real number, and *non-time-passage* actions, classified as *input* and *output* actions.
- $steps(A) : states(A) \times acts(A) \rightarrow states(A)$, a partial function that defines the possible steps (i.e., transitions).

Although in the general case, the transitions form a relation rather than a function, we have successfully reasoned about even nondeterministic timed automata using this definition by representing transitions with Hilbert's "choice" operator, ϵ.

There are two standard techniques for reasoning about properties of LV timed automata: proof by induction of state invariants and proof that one automaton simulates another. Ad hoc proofs concerning properties of execution sequences, such as timing properties, are also used.

3.2 Support for LV Timed Automata

A Template For LV Timed Automata. Our template for specifying LV timed automata in PVS provides a standard organization for an automaton definition. To define a particular LV timed automaton, the user supplies six items of information:

- declarations of the non-time actions,
- a type for the "basic state" (usually a record type) representing the state variables,
- any arbitrary state predicate that restricts the set of states (the default is **true**),
- the preconditions for all transitions,
- the effects of all transitions, and
- the set of start states.

In addition, the user may optionally supply

- declarations of important constants,
- an axiom listing any relations assumed among the constants, and
- any additional declarations or axioms desired.

Proof Strategies for Reasoning About LV Timed Automata. By providing an appropriately designed set of PVS strategies, TAME supports mechanical reasoning about timed automata using proof steps that mimic human proof steps.

One of the most important strategies provided by TAME for the LV timed automata model is the induction strategy that reduces induction proofs of state invariants to the point where only the nontrivial base and induction step cases remain. This strategy is based on a standard parameterized automaton theory called **machine**. Other strategies support the kind of reasoning typically needed in completing the proofs of these cases. Some examples: to reason about the arithmetic of time for time values that can be either non-negative real values or ∞, we have developed a special theory (called **time_thy**) and an associated simplification strategy (called **TIME_ETC_SIMP**); to simplify application of previously proved state invariants, we provide a strategy called **APPLY_INV_LEMMA**.

In addition, TAME has some initial support for the two other types of proofs that arise for timed automata, namely proofs of simulation, which are as highly structured as induction proofs, and ad hoc proofs with no particular fixed structure. For ad hoc proofs, we have identified a number of recurring types of reasoning that are (at least in principle) possible to automate so as

to resemble the corresponding human-sized reasoning step. We have implemented some of these, and have designs for others that will require some enhancements to PVS to implement. While a design for a simulation proof analog of the induction strategy exists, its implementation depends on improvements to PVS.

3.3 Current Usability of TAME

At present, using TAME requires the help of an expert in both deductive reasoning and PVS, even for induction proofs of state invariants. Besides needing some expertise to handle those invariants whose proof requires some excursions outside the set of strategies TAME provides, the user needs (among other things) an appropriate variant of the induction strategy for his or her application. The exact strategy variant needed for a particular timed automaton can, in principle, be compiled from certain details in the automaton specification. However, at present, this compilation must be done by hand. A general purpose strategy in PVS that probes for these details each time it is used could probably be created also; an open question is whether this would result in a significant loss of efficiency.

For an expert, TAME has proven to be a time-saver in entering specifications and checking their properties, and to provide useful feedback when errors are discovered.

3.4 Experiences With the Use of TAME

We have applied TAME to example specifications of real-time systems taken successively from [5], [10], [8], and [18]. TAME was originally designed to encode and to verify the specifications in [5]. The examples in [10] and [8] helped us to identify additional features needed in TAME to facilitate handling timed automata derived from MMT automata [13] and timed automata representing nondeterministic hybrid systems. The examples in [18] have been instructive exercises in identifying methods to support reasoning about nonlinear real arithmetic in PVS.

Applying TAME to one of the examples [8] exposed a number of specification and proof errors. Most were minor and easily fixed, but two were major. Our success in both detecting *and* correcting errors in this example demonstrates the utility of TAME. Errors were discovered when either a proof or type checking failed. Examining the context of the failure made it easy to discover the nature and location of the error and (in all but the most serious cases) to correct the error.

In every example we investigated, encoding new specifications using the TAME template has proved to be straightforward. The encoding of induction proofs by hand of state invariants into PVS proofs based on TAME's specialized strategies has resulted in PVS proofs that closely resemble the original hand proofs in a large number of cases. This number of cases should increase when improvements to PVS allow us to create smarter strategies. One example circumstance in which the resemblance currently breaks down slightly is when the state invariant to be proved has a complex logical structure—e.g., when it involves an embedded existential quantifier.

4 Future Plans for TAME

Specification and Proof Techniques To Be Supported. We plan to extend the capabilities of TAME in two directions. First, we expect to extend the set of specialized proof strategies for LV timed automata to cover proofs of simulation and a larger set of ad hoc proof steps. There are a number of such proof steps for which we have a design that can clearly be automated, but which cannot presently be automated in PVS. Certain planned enhancements to PVS will permit many of these steps to be implemented as PVS strategies. Second, we expect to add support for additional mathematical models that can be used to describe real-time systems. The next model we intend to support will be the SCR model [7].[1]

Enhancing PVS. PVS has several features that make it a good foundation for building specialized support. The expressiveness of its higher order logic and the extensiveness of its built-in decision procedures are two examples. But, as indicated above, some of the proof support we intend to supply with TAME requires enhancements to PVS. We have identified several enhancements that will provide TAME with better proof strategies. For example, appropriate assertion naming and tracking features can be used in solve the problem of embedded quantifiers mentioned in Section 3.4. The developers of PVS have undertaken to implement these enhancements, whose detailed requirements we are continuing to refine.

Interface Issues. In our experience with using TAME, many examples have arisen in which an interface outside of PVS would be useful in simplifying the interaction with PVS. Much simplification is possible because specification and proof are being done in a narrow context. For example, an interface could automate much of the creation of a specification consistent with a template, and enforce template conventions. An example of the help an interface could provide for proofs was mentioned in Section 3.3: compilation of appropriate versions of the induction strategy. For the near future, we will continue to study which services helpful to the user can be provided within PVS, and which ones are better provided outside of PVS.

5 Related Work

Other work exists that is similar in spirit to certain aspects of ours. Reference [16] describes the development of specialized theorem proving support based on PVS for another formal system, the duration calculus. Experience with making PVS itself more accessible to

[1] At present, timing has not been incorporated into this model. However, there are plans to do so. Moreover, the TAME methodology can in the meantime be applied to non-timed automata.

industrial users is reported in [17]. Another example in which the mechanization of hand proofs revealed errors is [9].

Most other work to mechanically verify real-time systems uses model checking. However, another effort closely related to ours uses a theorem-proving system, namely, the Larch Prover, to prove properties of LV timed automata [10]. The proofs include both induction proofs of state invariants and simulation proofs between automata. However, they do not include ad hoc style proofs, nor do they involve specialized proof techniques for timed automata.

References

[1] M. Archer and C. Heitmeyer. Mechanical verification of timed automata: A case study. In *Proc. 1996 IEEE Real-Time Technology and Applications Symp. (RTAS'96)*. IEEE Computer Society Press, 1996.

[2] Myla Archer and Constance Heitmeyer. Verifying hybrid systems modeled as timed automata: A case study. To be presented at HART'97, Grenoble, France, March, 1997.

[3] R. Boyer and J Moore. *A Computational Logic*. Academic Press, 1979.

[4] M. J. C. Gordon and T.F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic*. Cambridge University Press, 1993.

[5] C. Heitmeyer and N. Lynch. The Generalized Railroad Crossing: A case study in formal verification of real-time systems. In *Proc., Real-Time Systems Symp.*, San Juan, Puerto Rico, December 1994.

[6] C. Heitmeyer and D. Mandrioli, editors. *Formal Methods for Real-Time Computing*. Number 5 in Trends in Software. John Wiley & Sons, 1996.

[7] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. *ACM Trans. Software Eng. and Methodology*, July 1996.

[8] Gunter Leeb and Nancy Lynch. Proving safety properties of the Steam Boiler Controller: Formal methods for industrial applications: A case study. In Jean-Raymond Abrial, Egon Boerger, and Hans Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[9] P. Lincoln and J. Rushby. The formal verification of an algorithm for interactive consistency under a hybrid fault model. In C. Courcoubetis, editor, *Computer Aided Verification, CAV '93*, volume 697 of *Lecture Notes in Computer Science*, pages 292–304. Springer-Verlag, 1993.

[10] Victor Luchangco. Using simulation techniques to prove timing properties. Master's thesis, Massachusetts Institute of Technology, June 1995.

[11] N. Lynch and F. Vaandrager. Forward and backward simulations – Part II: Timing-based systems. To appear in *Information and Computation*.

[12] N. Lynch and F. Vaandrager. Forward and backward simulations for timing-based systems. In *Proc. of REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*, pages 397–446. Springer-Verlag, 1991.

[13] M. Merritt, F. Modugno, and M. R. Tuttle. Time constrained automata. In J. C. M. Baeten and J. F. Goote, editors, *CONCUR'91: 2nd Intern. Conference on Concurrency Theory*, volume 527 of *Lecture Notes in Computer Science*, pages 408–423. Springer-Verlag, 1991.

[14] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[15] N. Shankar, S. Owre, and J. Rushby. The PVS proof checker: A reference manual. Technical report, Computer Science Lab, SRI Intl., Menlo Park, CA, 1993.

[16] J. Skakkebaek and N. Shankar. Towards a duration calculus proof assistant in PVS. In *Third Intern. School and Symp. on Formal Techniques in Real Time and Fault Tolerant Systems, Lecture Notes in Computer Science 863*. Springer-Verlag, 1994.

[17] M. K. Srivas and S. P. Miller. Formal verification of a commercial microprocessor. Technical Report SRI-CSL-95-04, Computer Science Lab, SRI Intl., Menlo Park, CA, 1995.

[18] Henri Weinberg. Correctness of vehicle control systems: A case study. Master's thesis, Massachusetts Institute of Technology, February 1996.