Specifying and Proving Timing Properties with TIOA Tools

Dilsun Kaynar, Nancy Lynch, Sayan Mitra*
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar Street, Cambridge, MA 02141, USA.
{dilsun,lynch,mitras}@csail.mit.edu

Abstract

This paper introduces the TIOA specification language for timed systems, for example, communication protocols with timeouts or timing-sensitive distributed algorithms. TIOA specifications denote Timed Input/Output Automata, which are composable state machines that evolve using both discrete transitions and continuous trajectories. This paper also outlines a scheme for translating TIOA specifications to the language of the PVS theorem prover. A simple two-task race example illustrates the translation and the use of PVS in proving timing-dependent properties for timed I/O automata.

1. Introduction

The Timed Input/Output Automaton (TIOA) modeling framework is a basic mathematical framework that supports description and analysis of timed systems, for example, communication protocols with timeouts or timing-sensitive distributed algorithms. A timed I/O Automaton is a kind of nondeterministic, possibly infinite-state, state machine. The state of a TIOA is described by a valuation of state variables that are internal to the automaton. The state of a TIOA changes either instantaneously by the occurrence of a *discrete transition*, which is labeled by a discrete action, or according to a *trajectory*, which is a function that describes the evolution of the state variables over an interval of time. An important feature of the TIOA modeling framework is its support for compositional specification of timed systems. The framework defines what it means for one TIOA to *implement* another: A implements B if the external behavior set (set of traces) of A is contained in that of B. The framework also defines notions of *simulations*, such as forward simulations and backward simulations, which provide sufficient conditions for demonstrating implementation relationships. The theory of timed I/O automata is presented in detail in [4] and a shorter introduction to the framework appears in [5].

Recently, we have embarked on a project that will develop: (a) a formal modeling language called TIOA, (b) the front-end processor for TIOA, incorporating syntax and static semantic checking, and providing interfaces to computer-aided design tools, (c) a simulation tool allowing simulation of automata, (d) a theorem-proving link through an interface to the theorem-prover PVS [9], and (e) a suite of examples to illustrate the use of the tools for modeling and analyzing distributed algorithms, communication protocols, and other timing-based systems.

One line of our recent work focuses on proving timing-related properties of timed I/O automata using PVS. In our proofs we translate the example TIOA specifications to the input language of PVS by hand and use our PVS theory for timed I/O automata in doing the proofs. Working on examples in this fashion guides us in devising a translation scheme from TIOA to PVS, implementation of which will provide a TIOA to PVS interface. We also aim to identify recurring patterns in proofs so that we can build PVS strategies to mechanize these proofs as much as possible.

The described project builds upon the prior work on the IOA language [3] and TAME [1].

2. The TIOA Language

We use the `TwoTaskRace` automaton [10] given in Figure 2.1, to illustrate the TIOA language, its translation to PVS and a typical simulation proof. The signature, the state variable declarations and transition definitions in the TIOA language are similar to their counterparts in IOA [3]. Each variable in TIOA has an explicitly declared *static type* and an implicitly defined *dynamic type*. The built-in static types in TIOA extend those of IOA by the addition of a new type called `AugmentedReal`. This type is essentially the

* Research supported by DARPA/AFOSR MURI Contract F49620-02-1-0325 and ..

type Real extended with a constructor for infinity (∞). The dynamic type of any variable of static type Real or AugmentedReal is the set of piecewise continuous functions. The dynamic type of a variable whose type is neither Real nor AugmentedReal, or is **discrete** Real, is the set of piecewise constant functions. In other words, the value of such a variable remains constant throughout trajectories.

Each transition definition has a **pre**condition specifying when an action is enabled and an **eff**ect clause specifying the effects of making that transition.

The construct for specifying trajectories is a newly designed part of the TIOA language. Each trajectory definition defines a set of trajectories; the set of all trajectories for an automaton is the concatenation closure of all of these sets. A trajectory belongs to the set of trajectories defined by a trajectory definition if it satisfies the predicate in its **invariant** clause, the differential equations in the **evolve** clause and the stopping condition expressed by the **stop when** clause. The stopping condition is satisfied by a trajectory if the only state in which the condition holds is the last state of that trajectory. In other words, time cannot advance beyond the point where the stopping condition becomes true.

The automaton TwoTaskRace increments a counter until it performs a set action. After the occurence of a set action, it starts decrementing the counter and reports when the counter reaches 0. The variable now is used to keep track of realtime. The timing constraints are expressed using absolute times: (1) The variables firstmain and lastmain record, respectively, the first time and the last time that an action from the set {increment,decrement,report} is allowed to occur. (2) The variables firstset and lastset record, respectively, the first time and the last time that the action set is allowed to occur. There are two trajectory definitions corresponding to the two "modes" of the automaton. The trajectory definitions preset and postset specify, respectively, the trajectories of the automaton before and after performing the set action. The only continuous Real variable in this example is now, which increases at the constant rate 1 as expressed by the equation **d**(now)=1 in its **evolve** clause. The evolve statements in this example are very simple; however, the TIOA language permits a large class of differential and algebraic equations and inequalities.

In many timing-based systems, we are interested in finding lower and the upper bounds on the time of occurrence of some action. For example, for TwoTaskRace we want to prove that the upper bound on the time of occurrence of the report action is $b2 + a2 + b2(a2/a1)$. To show this we create an abstract automaton Abs which simply performs the report action within the aforementioned time. We then prove that TwoTaskRace implements Abs by showing the existence a forward simulation relation from TwoTaskRace to Abs. We specify the Abs automaton and the forward simulation relation in TIOA and

```
automaton TwoTaskRace(a1,a2,b1,b2 : Real)
signature
 internal increment, decrement, set
 output report

states
 count: Nat:=0, flag: Bool:=false,
 reported: Bool:=false, now: Real:=0,
 firstmain: discrete Real:=a1,
 lastmain: discrete AugmentedReal:=a2,
 firstset: discrete Real:=b1,
 lastset: discrete AugmentedReal:=b2

transitions
 internal increment      internal decrement
  pre ¬flag ∧             pre flag ∧
   now ≥ firstmain         now ≥ firstmain ∧
                           count > 0
  eff                     eff
   count:=count+1;         count:=count-1;
   firstmain:=now+a1;      firstmain:=now+a1;
   lastmain:=now+a2        lastmain:=now+a2

   internal set          output report
    pre ¬flag  ∧          pre flag ∧ count=0
     now ≥ firstset        ¬reported
                            ∧ now ≥ firstmain
    eff                   eff
     flag:=true;           reported:=true;
     firstset:=0;          firstmain:=0;
     lastset:=infty        lastmain:=infty

 trajectories
  trajdef preset          trajdef postset
   invariant  ¬flag        invariant flag
   stop when               stop when
    now = lastmain          now = lastmain
      ∨ now = lastset      evolve
   evolve d(now) = 1        d(now) = 1
```

Figure 2.1. TIOA code for TwoTaskRace

then translate them to PVS. Owing to shortage of space we omit these specifications from this paper; all the relevant files can be found at http://tioa.csail.mit.edu/project/example-pages/counters. Note also that the notion of a simulation relation is explained briefly in Section 1 and in more detail in [4] and [5].

3. Translation to PVS

Our approach for translating TIOA specifications to the PVS language uses the idea of TAME [1], which was developed for translating MMT automata [6] to PVS. The PVS

```
actions: DATATYPE                      states: TYPE =
  BEGIN                                  [# count: nat, flag: bool,
    ν(timeof: (fintime?)): nu?            reported: bool, firstmain: real,
    increment: increment?                 lastmain: time, firstset: real,
    decrement: decrement?                 lastset: time, now: time #]
    set: set?
    report: report?                     start(s: states): bool =
  END actions                             s = (# count := 0,flag := FALSE,
                                          reported := FALSE,firstmain := a1,
visible(a: actions): bool =               lastmain := a2, firstset := b1,
      report?(a) OR nu?(a)                lastset := b2, now := zero #)
```

Figure 3.2. PVS Spec for `TwoTaskRace`

specification of a TIOA consists of two parts: (a) auxiliary declarations, and (b) the PVS formulation of the automaton. The auxiliary declarations consist of theories describing the special data structures, such as stacks, timed-queues, and their related theorems that are used in the automaton's state but are not available in the PVS library. For example, the `AugmentedReal` type used in `TwoTaskRace` is not a standard PVS type and requires explicit definition. Since the type time included in the TAME library is identical to the TIOA type `AugmentedReal`, we use time instead of defining a new type for `AugmentedReal` in PVS. For a variable t of type time, if the value of t is finite, then we denote its real part is by $dur(t)$. The fintime type is a subtype of time consisting only of positive reals.

The second part of the PVS specification, which consists of components describing the TIOA, is described below.

Actions, State, and Transitions. To represent the actions of the TIOA, we define a PVS datatype called actions. This datatype consists of individual constructors for each action of the automaton (see Figure 3.2). To represent the trajectory definitions of TIOA we use the special *time-passage action* ν. The function visible declares the external actions of the automaton; it returns TRUE for the external actions. The third component is a type declaration for the states of the automaton; state is defined as a PVS record with different state components. The start predicate defines the start states of the automaton. The fifth and the final components are functions defining the precondition and the effect of each transition definition of the automaton (see Figure 3.3).

Trajectory Definitions. The trajectory definitions of the TIOA are captured in terms of special time-passage transition definitions. The `TwoTaskRace` automaton has two trajectory definitions with disjoint invariants and the same **evolve** clause. Each trajectory definition is deterministic in the sense that the state that is reached at the end of a given amount of time-passage is uniquely determined. We use a single transition definition ν in the PVS specification to represent these two trajectory definitions. As shown in Figure 3.3, the enabled predicate for $\nu(\text{delta\_t})$ specifies the stopping condition of the trajectory definitions. For example, the stopping conditions on the trajectory definitions of `TwoTaskRace` require time to stop whenever any deadline set by `lastmain` or `lastset` is reached. These stopping conditions are enforced by the two conjuncts in the precondition of $\nu(\text{delta\_t})$: $\text{now}(s) + \text{delta\_t} \leq \text{lastmain}(s)$, and $\text{now}(s) + \text{delta\_t} \leq \text{lastset}(s)$. The first condition states that delta_t units of time can advance from a given state only if the resulting time $\text{now}(s) + \text{delta\_t}$ does not exceed the $\text{lastmain}(s)$ deadline. Time-passage is enabled only if none of the disjuncts in the stopping conditions is violated.

The trans function gives the last state of a trajectory of length delta_t that results from solving the differential equations in the **evolve** part of the TIOA specification. This translation scheme assumes that there exists a unique closed form solution to the set of differential equations in the TIOA specification, and that the user can provide this solution. In the `TwoTaskRace` automaton, for example, the constant differential equations have a straightforward solution: for any trajectory τ, $\tau(t).now = \tau(0).now + t$. This solution is immediately translated in terms of delta_t in the transition definition for ν.

Although the trajectory definitions in this example are simple, our translation approach also works for more complex types of trajectories. For example, a set of overlapping trajectory definitions of a TIOA are specified by multiple time passage actions; each with its own enabling condition and transition definition. Trajectory definitions with nondeterminism in the **evolve** clause can be specified in PVS by adding extra parameters in the corresponding time-passage transition definition. For example, an **evolve** clause with **d**`(now) > 0` can be modeled by a time passage action ν(timeof: (fintime?), k:pos_real): nu? with two parameters, and the corresponding transition definition: $\nu(\text{delta\_t,k})$: s WITH [now := $\text{now}(s) + \text{k} * \text{delta\_t}$].

4. Forward Simulation Proof

Using this approach of translating TIOA specifications to PVS we construct PVS theories for the `Abs` and the `TwoTaskRace` automata. Recall that `Abs` is the abstract automaton that captures the essential timing properties of `TwoTaskRace`. Then we use the approach described in [7] to express a forward simulation relation from `TwoTaskRace` to `Abs` in a separate PVS theory. To show that the relation is indeed a forward simulation, we have to show that (a) for every start state of `TwoTaskRace` there is a start state of `Abs` that is related to it, and (b) every action of `TwoTaskRace` can be "matched by" a sequence of actions of `Abs` such that if the pre-states are related then the post states are also re-

enabled(a: actions, s: states): bool =
CASES a OF
ν(delta_t): delta_t $>$ zero $\wedge$ now(s) $+$ delta_t $\leq$ lastmain(s)
$\wedge$ now(s) $+$ delta_t $\leq$ lastset(s),
increment: ($\neg$ flag(s)) $\wedge$ dur(now(s)) $\geq$ firstmain(s),
set: ($\neg$ flag(s)) $\wedge$ dur(now(s)) $\geq$ firstset(s),
decrement: flag(s) $\wedge$ count(s) > 0
$\wedge$ dur(now(s)) $\geq$ firstmain(s),
report: flag(s) $\wedge$ count(s) $= 0$ $\wedge$ $\neg$ reported(s)
$\wedge$ dur(now(s)) $\geq$ firstmain(s)
ENDCASES

trans(a: actions, s: states): states =
CASES a OF
ν(delta_t): s WITH [now := now(s) $+$ delta_t],
increment: s WITH [count := count(s) $+ 1$,
firstmain := dur(now(s)) $+ a_1$, lastmain := now(s) $+ a_2$],
set: s WITH [flag := TRUE, firstset := 0, lastset := infinity],
decrement: s WITH [count := count(s) $- 1$,
firstmain := dur(now(s)) $+ a_1$, lastmain := now(s) $+ a_2$],
report: s WITH [reported := TRUE,
firstmain := 0, lastmain := infinity]
ENDCASES

Figure 3.3. PVS Spec for `TwoTaskRace` (cont.)

lated. Finding the matching sequence of actions is usually the key proof step. Note that in the TIOA framework, we use a separate case for time-passage steps in manual forward simulation proofs. In the PVS proofs, we deal only with actions since we represent time-passage in an automaton by a special time-passage action.

The proof of this forward simulation is carried out interactively in the PVS prover using some of the specialized strategies presented in [7]. These strategies break down the proof into the base case and the inductive cases for each action of `TwoTaskRace`. Most of the trivial subgoals generated at this stage are automatically discharged by the PVS decision procedure. Proofs of the non-trivial subgoals often rely on some simple invariant properties of the `TwoTaskRace` automaton most of which are proved automatically using TAME strategies and PVS decision procedures. Proving the simulation relation itself involves proving several inequalities involving real and time variables. We prove most of these inequalities by initially providing some guiding steps and then applying the strategies in the Field [8], and the Manip [2] packages, or the PVS decision procedures. Typical guiding steps include proving finiteness of the variables involved in the inequality or explicitly converting an expression of type time to the corresponding real expression.

5. Status of the TIOA Tools

The work presented in this paper was carried out within the scope of a larger tool development project described in the introduction. This project involves the integration of the front-end of the language with back-end tools such as the simulator and the theorem-prover PVS. Currently, the front-end of the TIOA language has been partially implemented. The simulator and the TIOA to PVS translator are in their design stage.

References

[1] M. Archer. TAME: PVS Strategies for special purpose theorem proving. *Annals of Mathematics and Artificial Intelligence*, 29(1/4), February 2001.

[2] Common Manipulations Ben. A pvs prover strategy package for.

[3] S. Garland, N. A. Lynch, J. Tauber, and M. Vaziri. *IOA User Guide and Reference Manual*. MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, 2003. Available at `http://theory.lcs.mit.edu/tds/ioa.html`.

[4] D. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. The theory of timed I/O automata. Technical Report MIT/LCS/TR-917a, MIT Laboratory for Computer Science, 2004. Available at `http://theory.lcs.mit.edu/tds/reflist.html`.

[5] D. K. Kaynar, N. A. Lynch, R. Segala, and F. Vaandrager. Timed I/O automata: A mathematical framework for modeling and analyzing real-time system. In *RTSS 2003: The 24th IEEE International Real-Time Systems Symposium*, Cancun,Mexico, December 2003.

[6] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In J. C. M. Baeten andJ. F. Goote, editor, *CONCUR '91: 2nd International Conference of Concurrency Theory*, volume 527, pages 408–423, 1991.

[7] S. Mitra and M. Archer. Reusable PVS proof strategies for proving abstraction properties of i/o automata. In *STRATEGIES 2004, IJCAR Workshop on strategies in automated deduction*, Cork, Ireland, July 2004.

[8] C. Muñoz and M. Mayero. Real automation in the field. Technical Report NASA/CR-2001-211271 Interim ICASE Report No. 39, ICASE-NASA Langley, ICASE Mail Stop 132C, NASA Langley Research Center, Hampton VA 23681-2199, USA, December 2001.

[9] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102 in Lecture Notes in Computer Science, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[10] A. Pnueli. Personal communication, 1988.