Developing Strategies for Specialized Theorem Proving about Untimed, Timed, and Hybrid I/O Automata *

Sayan Mitra[1] and Myla Archer[2]

[1] MIT Laboratory for Computer Science,
200 Technology Square, Cambridge, MA 02139
`mitras@theory.lcs.mit.edu`,
[2] Code 5546, Naval Research Laboratory,
Washington, DC 20375
`archer@itd.nrl.navy.mil`

Abstract. In this paper we discuss how we intend to develop a specialized theorem proving environment for the Hybrid I/O Automata (HIOA) framework [6] over the PVS [10] theorem prover, and some of the issues involved. In particular, we describe approaches to using PVS that allow and encourage the development of useful proof strategies, and note some desired PVS features that would further help us to do so for our HIOA environment.

1 Introduction

Interest in specialized theorem proving environments has emerged from various application domains [3, 12, 5, 1]. A major motivation for developing such environments is to relieve the developers and verification engineers from mastering the specification language and the proof commands of a general theorem prover. Specialized environments also help expert users of theorem provers by replacing repetitive proof patterns with strategies, and by making it possible to generate human readable proofs.

We plan to develop a specialized theorem proving environment to be used with the Hybrid I/O Automata (HIOA) framework. HIOA is a very general framework for modeling systems with both discrete and continuous behavior, and subsumes both the timed and untimed I/O automata models. Therefore, any strategies and metatheories for HIOA would be applicable to timed and untimed I/O automata as well. A theory template for specifying HIOAs has been presented in [8]. This formalization of HIOA in PVS is similar to the formalization of Lynch-Vaandrager (LV) timed automaton [7] in the Timed Automaton Modeling Environment (TAME) [1]. However, important differences arise in the two formalizations because LV-timed automata communicate via shared actions alone, whereas HIOAs also communicate via shared variables. Therefore, the evolution of continuous variables is modeled in TAME using time passage actions to

* Funding for this research has been provided by ONR

capture cumulative changes over an interval, while in the HIOA model, the evolution of the continuous state variables over time is modeled using trajectories. Our HIOA environment must allow for these differences.

The rest of this paper is organized as follows: In Section 2 we discuss the main types of proofs which will be supported by our HIOA environment and the design issues involved in developing proof strategies for each type. In Section 3 we suggest certain new features of PVS which would aid the development of strategies for PVS. Finally, we summarize and conclude in Section 4.

2 Supported Proof Types

Apart from simplifying direct proofs of properties, the HIOA proving environment will provide special strategies for mechanizing inductive invariant proofs and abstraction (e.g., simulation) proofs for timed, hybrid and untimed I/O automata. Apart from TAME, another theorem proving environment has been developed, based on Isabelle, which mechanizes invariant proofs for I/O automata [9]. In [2], the authors present a simulation proof of a leader election protocol in PVS. However, we have not come across any work which addresses the development of strategies for mechanizing simulation proofs.

2.1 Inductive Proofs

The approach we intend to take for supporting inductive invariant proofs is derived from the Timed Automaton Modeling Environment (TAME) [1]. As in TAME, we will develop a parameterized theory `machine` which defines the reachable states of an automaton in terms of its states, initial states, actions and (in case of hybrid I/O automata) activities [8]. This theory will also establish the theorem that allows proving invariants inductively. We will also develop a general theory template which can be instantiated with particular state variables and actions (optionally, activities) to obtain an *automatonName*`_decls` theory describing the automaton. The *automatonName*`_decls` theory will import an instance of the theory `machine` with the declared states and actions as parameters. Instantiation of the theory `machine` defines reachability and the induction theorem for the particular automaton. All the invariants and the associated lemmas of an automaton will be collected and proved in a theory named *automatonName*`_inv`.

The advantages of this (TAME) approach are as follows: (1) It is possible to write generic strategies which work for all automata specified using the template. The strategies for induction are tailored for the defined automaton template, and are defined in the file `pvs-strategies`. Therefore, (2) the user can use the specialized environment from within the PVS system. Finally, (3) it is easy to generate human readable proofs using the generic strategies, provided that the strategies implement proof steps meaningful to humans.

A slightly different approach has been taken by the developers of DisCo [5, 4], where the PVS specification of the automaton is processed by a "generator" to produce the proof scripts. One advantage of this approach, due to the clearly defined interface between the theorem prover (PVS) and the specialized environment (DisCo), is that the generated proof scripts are relatively insensitive to the modifications of the internals of theorem prover commands and data structures.

However, we would like our strategies to be directly applicable to all automata specified with our template theory. The success of our approach does depend on access to the data-structures in the proof state maintained by PVS, and the consistency of the behavior of PVS proof commands. We discuss the PVS support necessary to achieve this in Section 3.

2.2 Abstraction Proofs

Given automata A and C, it is often useful for the purposes of verification to show that there exists an *abstraction relation* between them. Several kinds of abstraction relation, e.g., homomorphism, refinement, forward and backward simulation, etc., are described in the literature, and there may also be other such relations of interest.

Abstraction proofs can be performed directly by specifying both automata A and C, and the abstraction relation between them, within the same PVS theory. However, this approach makes it difficult to construct generic strategies for automating the proofs, and to use invariants which have been proved separately for the individual automata.

Instead, we intend to make use of PVS support for *theory parameters*, as follows. Two parameters `A` and `C` of the type `automaton` theory (Figure 1) can be passed as parameters to the theory `abstraction` (Figure 2), which also takes the abstraction relation `absrel` and the action map `actmap` as parameters. The the-

```
automaton: THEORY

BEGIN
   actions : TYPE+;
   visible (a:actions) : bool;
   stutter? (a:actions) : bool;

   states : TYPE+;

   start (s:states) : bool;
   enabled (a:actions, s:states) : bool;
   trans (a:actions, s:states) : states;

   reachable (s:states) : bool;
   equivalent (s1, s2: states) : bool;
END automaton
```

Fig. 1. The `automaton` abstract theory

ory abstraction, which somewhat resembles the theory group_homomorphism in [11] for setting up proofs of homomorphism between groups, defines the abstraction relations between the two interpretations of the automaton theory. To pass actual theory parameters to group_homomorphism, the various elements of the group theories must be named: the members of the groups, identities and composition operators, etc. But, when individual automata follow the same naming conventions as in the theory automaton, a shortcut is in principle possible in passing actual theory parameters to abstraction: because the various elements of the actual parameters can be matched to the formal parameters *syntactically*, only the actual theory names need to be provided. A modification to PVS that will allow this shortcut is under construction at SRI.

```
abstraction [ A, C: automaton,
              actmap: [C.actions -> A.actions],
              absrel: [C.states, A.states -> bool] ] : THEORY
  BEGIN
    a_C : VAR C.actions;
    a_A : VAR A.actions;
    s_C, s1_C, s2_C: VAR C.states;
    s_A : VAR A.states;

    vis_ax: AXOIM
      (FORALL a_C: C.visible(a_C) => A.visible(actmap(a_C)));

    invis_ax: AXIOM
      (FORALL a_C: NOT(C.visible(a_C)) => A.stutter(actmap(a_C)));

    refinement_base : bool =
      (FORALL s_C, s_A:
         C.start(s_C) & absrel(s_C, s_A)
          => A.start(s_A));

    refinement_step : bool =
      (FORALL s_C, s1_C, a_C, s_A:
         C.reachable(s_C) &
         C.equivalent(s_C, s1_C) & C.visible(a_C) & C.enabled(a_C, s1_C) &
         A.reachable(s_A) &
         absrel(s1_C, s_A)
          => A.enabled(actmap(a_C), s_A) &
             (EXISTS (s2_C: C.states):
                C.equivalent(C.trans(a_C, s1_C), s2_C) &
                absrel(s2_C, A.trans(actmap(a_C), s_A))));

    refinement : bool = refinement_base & refinement_step;
  END abstraction
```

Fig. 2. The abstraction theory

The `actmap` relation in the theory `abstraction` maps an action of the concrete automaton C to an action of the abstract automaton A. The axioms `vis_ax` and `invis_ax` that indicate that the visible actions in C map to visible actions in A and invisible (i.e., internal) actions in C map to a stutter step in A, become proof obligations when `abstraction` is instantiated.

For abstraction proofs the theory `abstraction` assumes a role analogous to that of the theory `machine` in the case of induction proofs, in that it will define the abstraction relations and also establish the theorems (e.g., concerning trace inclusion) that are the consequences of the existence of such relations between pairs of automata. In Figure 2, only one sort of refinement relation has been defined; in practice, the theory `abstraction` will define all possible useful abstraction relations between the two automata. The theory `abstraction` will thus provide us with a starting point for developing generic strategies for proving abstraction relations.

3 PVS Support

In this section we suggest some PVS features which would be helpful for writing strategies, particularly for the above types of proofs.

1. **Naming in theory interpretations.** The abstraction proofs involve many related theories, for example different instances of *automatonName*`_decl`, *automatonName*`_inv`, `machine`, etc. It is difficult to write general strategies that involve formulas or definitions in multiple theories: the user often has to identify the particular theory instances explicitly. It would be useful for strategy writers if PVS provided well documented naming conventions and functions for determining theory instances associated with names, and supported the automatic context-based selection by user strategies of appropriate theory instances for names.
2. **Functions to access information in specification and in proof states.** A strategy often depends on the nature of the automaton specification. It can also make choices based on the current proof state. The CLOS structure used by PVS provides functions to access various slots of the current proof state object. However, these are not guaranteed to be fixed, and indeed can sometimes change dynamically. For writing strategies it would be helpful if functions to access the definitions in a particular theory—for example the invariance lemmas or the action definitions—and functions for accessing parts of a sequent, formulae, etc, were provided as a part of a PVS strategy language.
3. **Documentation of implementation details in PVS proof commands.** The LISP/CLOS functions used in writing the internal PVS strategies (e.g., `induct`) are not well documented. Many of these functions, for example `typep`, `tc-eq`, can be useful for writing new strategies. Therefore, proper documentation of these functions would save effort and help new strategy writers learn the art.

4. **Improved support for maintaining compatibility with PVS.** The effects of some basic PVS commands (e.g. `SKOLEM, EXPAND`) have altered over PVS versions owing largely to changes in PVS's decision procedures and their use in conjunction with such basic steps. As a result, strategies developed for older versions of PVS do not always work in the newer PVS versions. Therefore, it is highly desirable to provide a feature in future versions of PVS that would allow strategies to invoke prover commands and get the same result as in some specified previous version. Because most changes in effects appear to involve the decision procedures and their hidden uses, there should at the very least be optional versions of proof steps that decouple them from any use of these procedures.

4 Conclusions

Domain specific theorem proving is a practical means for harnessing the power of mechanical theorem provers for system design and testing. In this paper we have outlined design principles for the development of proof strategies of a specialized theorem proving environment for hybrid I/O automata based on PVS. Our aim is to make the more complex component of the environment—the proof strategies—generic, based on a specific HIOA template, leaving the simpler component—the specification—to be written by instantiating the template. We have outlined the support we believe would help us develop effective generic strategies.

Acknowledgements

We wish to thank John Rushby and Natarajan Shankar of SRI for helpful discussions about our plans for a framework supporting generic strategies for abstraction relations between automata. We thank Sam Owre and Natarajan Shankar for undertaking enhancements to PVS that will support our plans. We also thank Nancy Lynch of MIT for helpful discussions and her comments about the design of the specification language for HIOA.

References

1. Myla Archer. TAME: PVS Strategies for special purpose theorem proving. *Annals of Mathematics and Artificial Intelligence*, 29(1/4), February 2001.
2. M. Devillers, D. Griffioen, J. Romijn, and F. Vaandrager. Verification of a leader election protocol—formal methods applied to IEEE 1394. *Formal Methods in System Design*, 16(3):307–320, June 2000.
3. Urban Engberg. *Reasoning in the Temporal Logic of Actions - The Design and Implementation of an Interactive Computer System.* PhD thesis, University of Aarhus, Denmark, 1995.
4. Pertti Kellomäki. Mechanizing invariant proofs of joint action systems. In *Proceedings of the Fourth Symposium on Programming Languages and Software Tool*, pages 141–152, Visegrad, Hungary, June 1995.

5. Pertti Kellomäki. Mechanical verification of DisCo specifications. In *Israeli-Finnish Binational Symposium on Specification, Development, and Verification of Concurrent Systems*, Technion, Haifa, January 1996.
6. Nancy Lynch, Roberto Segala, and Frits Vaandraager. Hybrid I/O automata. To appear in *Information and Computation*. Also, Technical Report MIT-LCS-TR-827d, MIT Laboratory for Computer Science Technical Report, Cambridge, MA 02139, January 13, 2003.
`theory.lcs.mit.edu/tds/papers/Lynch/HIOA-final.ps`.
7. Nancy Lynch and Frits Vaandrager. Forward and backward simulations - part ii: Timing-based systiems. *Information and Computation*, 128(1):1–25, July 1996.
8. Sayan Mitra. HIOA+: Specification language and proof tools for hybrid systems, 2003. Submitted for publication, http://theory.lcs.mit.edu/ mitras/research/LCPTHIOA.ps.
9. Olaf Müller. *A Verification Environment for I/O Automata Based on Formalized Meta-Theory*. PhD thesis, Technische Universität München, Sept. 1998.
10. 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.
11. S. Owre and N. Shankar. Theory interpretations in PVS. Technical report, Computer Science Lab., SRI Intl., Menlo Park, CA, 2001.
12. S. Kalvala. A Formulation of TLA in Isabelle. In E.T. Schubert, P.J. Windley, and J. Alves-Foss, editors, *8th International Workshop on Higher Order Logic Theorem Proving and its Applications*, volume 971, pages 214–228, Aspen Grove, Utah, USA, 1995. Springer-Verlag.