In E. A. Emerson and A. P. Sistla, editors, Computer-Aided Verification, CAV '2000, volume 1855 of Lecture Notes in Computer Science, pages 548-551, Chicago, IL, July 2000. ©Springer-Verlag

Integrating WS1S with PVS*

Sam Owre and Harald Rueß

Computer Science Laboratory
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025, USA
{owre, ruess}@csl.sri.com

There is a growing trend to integrate theorem proving systems with specialized decision procedures and model checking systems. The proving capabilities of the PVS theorem prover, for example, have been improved considerably by extending it with new proof tactics based on a BDD package, a μ-calculus model-checker [4], and a polyhedral library. In this way, a theorem proving system like PVS provides a common front-end and specification language for a variety of specialized tools. This makes it possible to use a whole arsenal of verification and validation methods in a seamless way, combine them using a strategy language, and provide development chain analysis.

Here we describe a novel PVS tactic for deciding an interesting fragment of PVS that corresponds to the *Weak Second-order Theory of 1 Successor*, WS1S. This logic may not only be viewed as a highly succinct alternative to the use of regular expressions, but can also be used to encode Presburger arithmetic or quantified boolean logic. The decidability of WS1S is based on the fact that regular languages may be characterized by logics. However, this automata-theoretic procedure is of staggering complexity, namely non-elementary.

Although this logic-automaton connection has been known for more than 40 years, it was only through the recent work at BRICS that it become possible to make effective use of automata-based decision procedures for logics like WS1S. Their tool, called MONA [2], acts as a decision procedure and as a translator to finite-state automata. It is based on new algorithms for minimizing finite-state automata using binary decision diagrams (BDDs) to represent transition functions in compressed form. Various applications of MONA—including hardware verification, validation of software design constraints, and establishing safety and liveness conditions for distributed systems—are mentioned in [2].

We are using the efficient automata-construction capabilities of MONA for building a tactic that decides a fragment of the PVS specification language. This fragment includes boolean expressions, arithmetic on the natural numbers restricted to addition/subtraction with/from a constant, and operations on finite sets over the naturals like union, intersection, set difference, addition and removal of a natural. Predicates include arithmetic comparisons, equality, disequality, the subset relation, and membership in the form of function application `P(i)`.

* This research was funded by DARPA AO D855 under US Air Force Rome Laboratory contract F30602-96-C-0204 and by the National Science Foundation Contract No. CCR-9712383.

Moreover, there is quantification over the booleans, the natural numbers, finite sets of naturals, and predicate subtypes of the aforementioned types built from WS1S formulas. Finite sets of natural numbers may also be described using the choice operator `the`. In this way, ripple-carry addition may be defined as follows.

```
+(P, Q: finite_set[nat]): finite_set[nat] =                               1
  the({R | EXISTS (C: finite_set[nat]): NOT(C(0)) AND
             FORALL (t: nat):
               (C(t+1) = ((P(t)&Q(t)) OR (P(t)&C(t)) OR (Q(t)&C(t)))) &
                 (R(t) = P(t) = Q(t) = C(t))});
```

Here a natural number `k` is mapped to a set of indices corresponding to `1`'s in the binary representation of `k`; e.g., `10` is represented as {`1`,`3`}. Usually, functions are encoded in a relational style in WS1S, but the inclusion of the choice operator allows one to provide functional representation. Notice that the carry vector `C` is hidden by means of second-order quantification.

This fragment of the PVS language is being decided by associating an automata with every formula. This translation proceeds in two steps. First, definitions are unfolded to transform formulas into the language of WS1S. In this step, the full arsenal of theorem proving capabilities of PVS, including decision procedures, rewriting, and lemma application, may be used to simplify the resulting formula. The second step of the translation traverses this formula and recursively builds up a corresponding automata that recognizes the language of interpretations of the PVS formula. Here we use foreign function calls to directly call the C functions of the MONA library [2] from the Lisp implementation of PVS.

Moreover, we use facilities provided by the Allegro Lisp garbage collector to also take the automaton and BDD data structures of MONA into account. In this way, we create a transparent and functional view of the MONA capabilities. This makes it possible, for example, to memoize the translation process. In addition, the translation of PVS formulas to deterministic finite automata includes abstraction. Whenever the translator encounters a PVS expression outside the scope of WS1S, it creates a new variable—of type `nat`, `bool`, or `finite_set[nat]`, depending on the type of the expression—and replaces the original expression with this variable throughout the formula. This abstraction increases the scope of our proof procedure. When the expression to be abstracted away includes bound variables, however, the translator gives up. Abstraction has been found particularly useful in the analysis of state machines, where the state usually consists of a record of state components and the specification includes accesses to these components that can easily be abstracted. This translation has been packaged as a new PVS tactic.

```
  stamps: LEMMA                                                           2
    FORALL (i: finite_set[nat]): i >= [| 8 |] =>
      EXISTS (j, k: finite_set[nat]): i = 3 * j + 5 * k
```

```
ltl: THEORY
BEGIN
   time : TYPE = nat;
   tpred: TYPE = finite_set[time]

   t, t1  : VAR time
   P, Q, R: VAR tpred

   [](P)(t): boolean = FORALL (t1: upfrom[t]): P(t1);
   <>(P)(t): boolean = EXISTS (t1: upfrom[t]): P(t1);

   |=(P): boolean = P(0);          % Validity

   WF(A, EA, I: tpred): tpred = ([]<>(A & NOT(I))) OR []<>(NOT(EA));
   SF(A, EA, I: tpred): tpred = ([]<>(A & NOT(I))) OR <>[](NOT(EA));
END ltl
```

Fig. 1. Encoding of a Linear Temporal Logic

Consider, for example, the Presburger formula `stamps`, where `*` is defined by iterating addition as defined above, and `[| k |]` is a recursively defined function for computing a finite set that represents the unsigned representation of `k`.[1] This inductive property is shown to be valid by simply calling the `(ws1s)` tactic within the PVS prover, since, after unfolding and unwinding all definitions, including the recursive ones, the associated automata recognizes the full language of interpretations. Most proof attempts, however, fail.

```
    (EXISTS (x: nat): x = 100 & P!1(x))                          3
  & (EXISTS (x: nat): x = 111 & P!1(x))
  & (FORALL (i: nat): i /= 100 & i /= 111 => NOT P!1(i))
```

This formula is neither valid nor unsatisfiable. Thus, the tactic `(ws1s)` returns with a counterexample (`P!1 = emptyset[nat]`) and a witness (`P!1 = add(111, add(100, emptyset[nat]))`) for the free variable `P!1`.

Using the approach outlined above we have encoded various theories in WS1S: Presburger arithmetic, lossy queues, regular expressions, a restricted linear temporal logic (LTL), and fixed-sized bitvectors. The complete encoding of a LTL, including definitions for weak and strong fairness is shown in Figure 1. Boolean connectives do not have to be defined explicitly for this logic, since the conversion mechanism of PVS [3] automatically lifts the built-in logic connectors to the type `tpred`. Regular expressions are represented by means of a datatype, and a recursively defined meaning function associates the WS1S defined set of

[1] Notice that `[| k |]` is not a WS1S arithmetic relation, because otherwise one could define addition directly on the natural numbers; but this is outside the scope of WS1S.

words recognized by this regular expression; hereby, a word is of type `[below[N] -> finite_set[nat]]`, where `N` specifies an alphabet of size `expt(2,N)`.

Notice that we automatically have a decision procedure for the combination of the encoded theories mentioned above, since all encodings are definitional extensions of the base language of WS1S. Moreover, the list of encoded theories is open-ended.

Among other applications we have been using the `(ws1s)` tactic for discharging verification conditions that have been generated using the abstraction method described in [1]. These examples usually involve a combination of the lossy queue theory and regular expressions. Also, quantifier reasoning and Skolemization has to be used to transform the formulas under consideration into the fragment supported by the translation process. For example, universal-strength quantification over words needs Skolemization, since words are elements of a third-order type (see above). After preprocessing and unfolding the definitions, verification conditions are typically proved within a fraction of a second.

Altogether, the WS1S decision procedures enrich PVS by providing new automated proving capabilities, an alternative approach for combining decision procedures, and a method for generating counterexamples from failed proof attempts. On the other hand, this integration provides an appealing frontend and input language for the MONA tool, and permits using automata-theoretic decision procedures in conjunction with both traditional theorem proving techniques and specialized symbolic analysis like abstraction. This connection, however, needs further exploration.

The WS1S decision procedure has been integrated with PVS 2.3, which has been released in fall '99. PVS 2.3 is freely available at `pvs.csl.sri.com`.

Acknowledgements. We would like to thank A. Møller for clarifying discussions about MONA internals, M. Sorea for comments on this paper, and S. Bensalem for providing interesting test cases.

References

1. P.A. Abdulla, A. Annichini, S. Bensalem, A. Bouajjani, P. Habermehl, and Y. Lakhnech. Verification of infinite-state systems by combining abstraction and reachability analysis. volume 1633 of *Lecture Notes in Computer Science*, pages 146–159, Trento, Italy, July 1999. Springer-Verlag.
2. N. Klarlund and A. Møller. *MONA Version 1.3 User Manual*. BRICS Notes Series NS-98-3, Dept. of Computer Science, University of Aarhus, October 1998.
3. S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.
4. S. Rajan, N. Shankar, and M.K. Srivas. An integration of model-checking with automated proof checking. In Pierre Wolper, editor, *Computer-Aided Verification, CAV '95*, volume 939 of *Lecture Notes in Computer Science*, pages 84–97, Liege, Belgium, June 1995. Springer-Verlag.