8-2006

Formal Verification of Infinite State Systems Using Boolean Methods

Randal E. Bryant
Carnegie Mellon University

Formal Verification of Infinite State Systems Using Boolean Methods*

Randal E. Bryant

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA
Randy.Bryant@cs.cmu.edu

Most successful automated formal verification tools are based on a bit-level model of computation, where a set of Boolean state variables encodes the system state. Using powerful inference engines, such as Binary Decision Diagrams (BDDs) and Boolean satisfiability (SAT) checkers, symbolic model checkers and similar tools can analyze all possible behaviors of very large, finite-state systems.

For many hardware and software systems, we would like to go beyond bit-level models to handle systems that are truly infinite state, or that are better modeled as infinite-state systems. Examples include programs manipulating integer data, concurrency protocols involving arbitrary numbers of processes, and systems containing buffers where the sizes are described parametrically.

Historically, much of the effort in verifying such systems involved automated theorem provers, requiring considerable guidance and expertise on the part of the user. We would like to devise approaches for these more expressive system models that retain the desirable features of model checking, such as the high degree of automation and the ability to generate counterexamples.

We have developed UCLID [1], a prototype verifier for infinite-state systems. The UCLID modeling language extends that of SMV [9], a bit-level model checker, to include state variables that are integers, as well as functions mapping integers to integers and integers to Booleans. Functional state variables can be used to define array and memory structures, including arrays of identical processes, FIFO buffers, and content-addressable memories.

System operation is defined in UCLID in terms of the initial values and next-state functions of the state variables. Integer operations include linear arithmetic and relational operations. Functions can be defined using uninterpreted function symbols, as well as via a restricted form of lambda expression. The underlying logic is reasonably expressive, yet it still permits a decision procedure that translates the formula into propositional logic and then uses a SAT solver [7].

UCLID supports multiple forms of verification, requiring different levels of sophistication in the handling of quantifiers. All styles verify that a safety property of the form $\forall \mathcal{X} P(s)$ holds for some set of system states s, where $\mathcal{X}$ denotes a set of integer *index variables*. Index variables can be used to express universal properties for all elements in an array of identical processes, all entries in a FIFO buffer, etc.

The simplest form of *bounded property checking* allows the user to determine that property $\forall \mathcal{X} P(s)$ holds for all states reachable within a fixed number of steps k from an

* This research was supported by the Semiconductor Research Corporation, Contract RID 1029.001

initial state. Verifying such a property can be done by direct application of the decision procedure. In practice, the effort required to verify such a property grows exponentially in k, limiting the verification to around 10–20 steps. However, it provides a useful debugging tool. In our experience, most errors are detected by this approach.

Of course, it is important to verify that properties hold for all reachable states of the system. Unfortunately, the standard fixed-point methods for bit-level model checking do not work for infinite-state systems. In many cases, the system will not reach a fixed point within a bounded number of steps. Even for those that do, checking convergence is undecidable, and our efforts to implement incomplete methods for this task have had limited success [2].

To prove that property $\forall \mathcal{X} P(s)$ holds for all reachable states s, UCLID supports *inductive invariant* checking, where the user provides an invariant Q such that Q holds for all initial states, Q implies P, and any successor for a state satisfying Q must also satisfy Q. This latter condition requires proving the validity of a formula containing existentially quantified index variables. Although this problem is undecidable for our logic, we have successfully implemented an incomplete approach using quantifier instantiation [8].

A more automated technique is to derive an inductive invariant via *predicate abstraction* [4]. Predicate abstraction operates much like the fixed-point methods of symbolic model checking, but using the concretization and abstraction operations of abstract interpretation [3] on each step. We have generalized predicate abstraction to handle the indexed predicates supported by UCLID [6]. Each step requires quantifier elimination to eliminate the current state variables, much like the relational product step of symbolic model checking. We implement this step by performing SAT enumeration on the translated Boolean formula.

As a final level of automation, we can automatically discover a set of relevant predicates for predicate abstraction based on the property P and the next-state expressions for the state variables [5].

We have successfully verified a number of systems with UCLID, including out-of-order microprocessors, distributed cache protocols, and distributed synchronization protocols.

References

1. R. E. Bryant, S. K. Lahiri, and S. A. Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In E. Brinksma and K. G. Larsen, editors, *Computer-Aided Verification (CAV '02)*, LNCS 2404, pages 78–92, 2002.
2. R. E. Bryant, S. K. Lahiri, and S. A. Seshia. Convergence testing in term-level bounded model checking. In *Correct Hardware Design and Verification Methods (CHARME '03)*, LNCS, September 2003.
3. P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for the static analysis of programs by construction or approximation of fixpoints. In *Principles of Programming Languages (POPL '77)*, pages 238–252, 1977.
4. S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *Computer-Aided Verification (CAV '97)*, LNCS 1254, pages 72–83, 1997.

5. S. K. Lahiri and R. E. Bryant. Indexed predicate discovery for unbounded system verification. In *Computer-Aided Verification (CAV '04)*, LNCS 3114, pages 135–147, 2004.
6. S. K. Lahiri and R. E. Bryant. Indexed predicate abstraction. *ACM Transactions on Computational Logic*, To appear.
7. S. K. Lahiri and S. A. Seshia. The UCLID decision procedure. In *Computer-Aided Verification (CAV '04)*, LNCS 3114, pages 475–478, 2004.
8. S. K. Lahiri, S. A. Seshia, and R. E. Bryant. Modeling and verification of out-of-order microprocessors in UCLID. In M. D. Aagaard and J. W. O'Leary, editors, *Formal Methods in Computer-Aided Design (FMCAD '02)*, LNCS 2517, pages 142–159, 2002.
9. K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1992.