System Modeling and Verification with UCLID

Randal E. Bryant
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA 15213 USA
Randy.Bryant@cs.cmu.edu

Formal verification has had a significant impact on the semiconductor industry, particularly for companies that can devote significant resources to creating and deploying internally developed verification tools. If we look more closely, however, we see that the major industrial applications of formal verification have been either in verifying individual blocks, such as floating-point units and memories, or in verifying an abstracted representation of some aspect of the system, such as a cache coherence protocol. Attempting to verify overall system correctness is beyond the reach of current tools. For example, these tools are not capable of verifying that an out-of-order execution microprocessor correctly replicates the behavior of its sequential instruction set architecture (ISA) model.

Most existing verification tools model system operation at a detailed bit level. Using powerful inference engines, such as Binary Decision Diagrams (BDDs) and Boolean satisfiability (SAT) checkers, symbolic model checkers [3, 5] and similar tools can analyze all possible behaviors of very large, finite-state systems. Modeling a system at the bit level makes it difficult to scale formal verification to systems that store and manipulate large amounts of data, such as microprocessors and many forms of embedded software. The many bits of state held in the various memories, queues, and caches lead to state explosion problems that overwhelm even the most advanced model checkers.

Taking a cue from the hardware design principle of separating data from control, we believe that systems should be modeled and verified using a more abstract representation of data. If we assume individual functional units, such as ALUs and instruction decoders can be verified separately, then there is no need to track the value of every bit in the system. Instead, we can represent words of data as symbolic *term* values that are generated by function units, communicated among different subunits, and stored in different buffers and memories. For some term value x, we need not keep track of its bit width, its encoding, or even its actual value. With this *term-level* abstraction of data, verification can focus on the behavior of the control logic.

Term-level abstraction has long been used by researchers using automatic theorem provers to verify hardware design [10]. Burch and Dill [4] were among the first to demonstrate that highly automated tools based on term-level models could be used to verify pipelined microprocessors. Rather than using Boolean inference engines, these tools make use of decision procedures for highly restricted subsets of first-order logic. Over the years, these procedures have improved greatly in their speed [11] and the richness of the logic they can handle [1, 6].

We have developed UCLID [2], a prototype verifier for infinite-state systems. The UCLID modeling language extends that of SMV, a bit-level model checker, to include integer and function state variables, addition by constants, and relational operations. The underlying logic is expressive enough to model a wide range of systems, but it still permits a decision procedure where we transform the formula into propositional logic and then use either BDDs or a SAT solver. Most recently, we have developed powerful predicate abstraction methods that can automatically generate and prove system invariants using techniques similar those used in symbolic model checking [8]. UCLID has been used to verify a variety of hardware designs, including out-of-order microprocessors [7] and cache coherency protocols, as well as abstract synchronization protocols such as Lamport's Bakery algorithm [9].

References

[1] C. Barrett, D. Dill, and A. Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In E. Brinksma and K. G. Larsen, editors, *Computer-Aided Verification (CAV '02)*, LNCS 2404, pages 236–249, 2002.

[2] 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.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential circuit verification using symbolic model check-

ing. In *27th Design Automation Conference (DAC '90)*, pages 46–51, 1990.

[4] J. R. Burch and D. L. Dill. Automated verification of pipelined microprocessor control. In D. L. Dill, editor, *Computer-Aided Verification (CAV '94)*, LNCS 818, pages 68–80, 1994.

[5] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[6] J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated Canonizer and Solver. In G. Berry, H. Comon, and A. Finkel, editors, *Computer-Aided Verification (CAV '01)*, LNCS 2102, pages 246–249, 2001.

[7] S. K. Lahiri and R. E. Bryant. Deductive verification of advanced out-of-order microprocessors. In *Computer-Aided Verification (CAV '03)*, LNCS 2725, pages 341–354, 2003.

[8] S. K. Lahiri, R. E. Bryant, and B. Cook. A symbolic approach to predicate abstraction. In W. A. Hunt, Jr. and F. Somenzi, editors, *Computer-Aided Verification (CAV '03)*, LNCS 2725, pages 141–153, 2003.

[9] L. Lamport. A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17:453–455, August 1974.

[10] J. Sawada and W. A. Hunt, Jr. Processor verification with precise exceptions and speculative execution. In A. J. Hu and M. Y. Vardi, editors, *Computer-Aided Verification (CAV '98)*, LNCS 1427, pages 135–146, 1998.

[11] M. N. Velev and R. E. Bryant. Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors. In *38th Design Automation Conference (DAC '01)*, pages 226–231, 2001.