BAT: The Bit-Level Analysis Tool*

Panagiotis Manolios[1], Sudarshan K. Srinivasan[2], and Daron Vroon[1]

[1] College of Computing
[2] School of Electrical & Computer Engineering
Georgia Institute of Technology
Atlanta, GA-30313
{manolios@cc.gatech.edu, darshan@ece.gatech.edu, vroon@cc.gatech.edu}

Abstract. While effective methods for bit-level verification of low-level properties exist, system-level properties that entail reasoning about a significant part of the design pose a major verification challenge. We present the Bit-level Analysis Tool (BAT), a state-of-the-art decision procedure for bit-level reasoning that implements a novel collection of techniques targeted towards enabling the verification of system-level properties. Key features of the BAT system are an expressive strongly-typed modeling and specification language, a fully automatic and efficient memory abstraction algorithm for extensional arrays, and a novel CNF generation algorithm. The BAT system can be used to automatically solve system-level RTL verification problems that were previously intractable, such as refinement-based verification of RTL-level pipelined machines.

1 Introduction

The Bit-level Analysis Tool (BAT) [5] is a system for verifying bit-level problems arising from hardware, software, and security domains. BAT implements a state-of-the-art decision procedure for solving quantifier-free formulas over the extensional theory of fixed-size bit-vectors and fixed-size bit-vector arrays (memories). BAT is a publicly available tool that can be downloaded from the BAT Webpage [5].

Our primary goal in developing BAT is to enable the verification of high-level properties of complex systems described at the bit-level, such as the verification of bit-level pipelined machine models. We have been able to use BAT to verify a 32-bit 5 stage pipelined machine in approximately 2 minutes [4]. Key features of BAT that enable the verification of complex systems such as pipelined machines are a fully automatic and efficient algorithm for abstracting bit-level memories [4] and a novel method for generating CNF (Conjunctive Normal Form) from a high-level circuit representation [6].

2 The BAT Specification Language

The BAT specification language is a strongly typed, Lisp-like language whose types include bit-vectors, bit-vector memories, and sequences over these types (multiple value types). We invested much effort in designing a powerful, general, and usable language

* This research was funded in part by NSF grants CCF-0429924, IIS-0417413, and CCF-0438871.

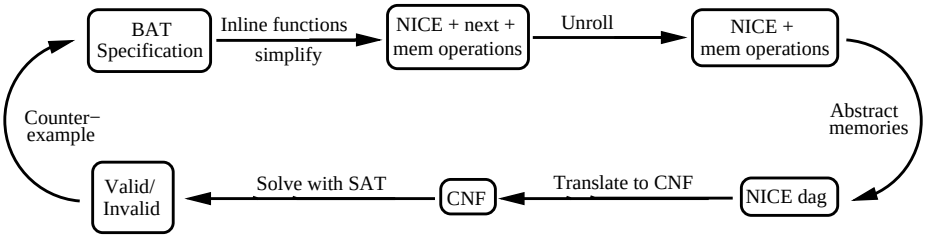

Fig. 1. The Bit-level Analysis Tool (BAT) decision procedure.

that can be the target for synthesizable subsets of VHDL or Verilog. The language also allows for a clear separation of concerns between models and specifications, a feature that drastically simplifies the effort required to describe system-level properties of complex hardware models such as pipelined machines.

An important feature of the BAT language is that memories are treated as first class objects. Memories can be compared for equality and inequality in all contexts, and they can be passed as arguments to functions and returned by functions. Other key features of the language include a type inference algorithm that can determine the type of any expression; this alone finds many silly mistakes, especially when one is experimenting with parametrized modes. The language allows users to define functions that can be used not only to model systems, but to specify their correctness. BAT also provides multiple value types, *i.e.*, users can create a sequence of any types. Finally, BAT provides support for easily defining parametrized models.

The BAT language supports a variety of bit-vector operations including bit-vector comparisons such as equality, less than, and greater than; various types of shift operations; arithmetic operations including modular and machine addition, subtraction, and multiplication; bitwise operations such as bitwise conjunction, disjunction, negation, implication. The language also supports temporal operators AG and AF. Applicative functions for reading and updating memories are also supported.

3 The BAT Decision Procedure

BAT can be used as a decision procedure, a bounded model checker, or as a k-induction engine. BAT takes a specification described using the BAT language as input, and compiles this specification to a SAT problem in four high-level steps. The SAT problem is then checked with a SAT solver. If a bug is found, BAT generates a counterexample in terms of the original BAT specification. The compilation to CNF is performed using a novel data structure for representing circuits, known as the NICE dag, because it contains Negations, Ites, Conjunctions, and Equivalences.

We now describe the four high-level compilation steps. First, BAT inlines functions, propagates constants, and performs a simplification step to transform the original specification to a NICE dag extended with *next* operators (used to specify the transition relation) and memory operators. Second, the transition relation is unrolled, resulting in a NICE dag extended with memory operations. Third, the memories are abstracted using BAT's memory reduction algorithm and the memory operations for the resulting

reduced memories are replaced with their equivalent Boolean circuits, resulting in a NICE dag. Fourth, the NICE dag is translated to a SAT problem in CNF. Note that BAT can make a decision after any of the above steps. For example, during simplification, BAT may simplify the problem to true or false.

3.1 Memory Abstraction

BAT implements a sound, complete, fully automatic, and efficient memory abstraction algorithm that can deal with an extensional theory of finite bit-vector memories [4]. The use of memory abstraction is crucial in bit-level verification problems as the presence of large memories would otherwise lead to intractable SAT problems.

The key idea of the BAT memory abstraction algorithm is to reduce memories to manageable sizes in a sound and complete way. This is possible because, even for very large memories, correctness conditions tend to refer to only a relatively small collection of memory references, which can be to any part of memory, however.

The complexity of the resulting verification problem depends heavily on the size of the abstracted memory, which is based on the number of unique memory accesses. Our memory abstraction algorithm includes term-rewriting techniques that are very effective in simplifying expressions containing memory operations. This allows us to recognize when syntactically distinct expressions correspond to the same memory address, which leads to more efficient memory abstractions and eventually to simpler SAT problems.

We deal with extensionality by keeping the abstract memories around. The intuition is that this allows us to compare memories for equality or inequality by comparing the abstract memories directly. To make this sound and efficient, a more sophisticated analysis is required, which is presented in our previous work on memory abstraction [4].

3.2 Efficient Translation to CNF

BAT uses a novel and efficient approach to generate SAT problems based on the use of NICE dags, a new data structure for representing circuits [6]. This is an important problem because, while modern SAT solvers have become proficient at solving Boolean satisfiability problems in CNF, these problems mostly arise from general Boolean circuits that are then translated to CNF. Furthermore, the CNF translation algorithm can significantly impact verification times. Experimental evaluation based on over 8,000 benchmarks showed that our CNF generation algorithm leads to significant time savings over both the widely used Tseitin algorithm [7] and Jackson and Sheridan's state-of-the-art algorithm [2]. For example, Minisat2 was able to handle all the SAT problems generated by the BAT CNF translation algorithm, whereas, it timed out on many of the SAT problems generated by both the Tseitin and Jackson/Sheridan algorithms.

4 Applications

BAT is the first bit-level reasoning tool that has been used successfully to verify non-trivial bit-level pipelined machines automatically [4]. Pipelined machine verification entails showing that the pipelined machine refines its instruction set architecture. This is a computationally demanding problem as it requires reasoning about a large part of

the control logic of the system. The problem also involves comparing large memories for equality. With previous work, it was only possible to automatically solve pipelined machine verification problems at the term-level.

Using BAT we have been able to prove automatically that a 32-bit 5 stage pipelined machine refines its ISA in about 125 seconds. Other state-of-the-art tools that we have tried, including Yices [1] (winner of the 2006 SMT competition), cannot solve simple 2-stage pipelined machine problems. Such problems can be solved with BAT in less than a second. We have also been able to use BAT in a compositional verification flow to check complex pipelined machines with as many as 10 stages [3].

5 Conclusions

We have presented the Bit-level Analysis Tool (BAT), a state-of-the-art decision procedure for bit-level reasoning. We have used BAT to solve system-level verification problems, including the verification of pipelined machine which cannot be handled by other verification tools. For example, BAT can be used to verify that a 32-bit 5-stage pipelined machine model refines its instruction set architecture in approximately 2 minutes. The BAT language is feature-rich and enables users to effectively model systems and to specify properties. From an algorithm point of view, we described two key advances that are implemented in BAT. One is an efficient, automatic, sound, and complete memory abstraction algorithm for extensional arrays that is further improved with term-rewriting techniques. The second is a novel circuit to CNF conversion algorithm that provides significant improvements over other available CNF conversion algorithms. For future work, we plan to extend BAT with a counterexample guided abstraction-refinement framework, explore the use of more advanced term-rewriting techniques, and consider methods for automatically abstracting data paths.

References

[1] B. Dutertre and L. M. de Moura. A fast linear-arithmetic solver for DPLL(T). In T. Ball and R. B. Jones, editors, *Computer Aided Verification, CAV 2006*, pages 81–94, 2006.

[2] P. Jackson and D. Sheridan. Clause form conversions for boolean circuits. In H. H. Hoos and D. G. Mitchell, editors, *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004*, volume 3542 of *LNCS*, pages 183–198. Springer, 2004.

[3] P. Manolios and S. K. Srinivasan. A complete compositional reasoning framework for the efficient verification of pipelined machines. In *International Conference on Computer-Aided Design (ICCAD'05)*, pages 863–870. IEEE Computer Society, 2005.

[4] P. Manolios, S. K. Srinivasan, and D. Vroon. Automatic memory reductions for RTL-level verification. In *ICCAD 2006, ACM-IEEE International Conference on Computer Aided Design*. ACM, 2006.

[5] P. Manolios, S. K. Srinivasan, and D. Vroon. BAT: The Bit-level Analysis Tool. 2006. Available from `http://www.cc.gatech.edu/~manolios/bat/`.

[6] P. Manolios and D. Vroon. Efficient circuit to CNF conversion. In *International Conference on Theory and Applications of Satisfiability Testing*, 2007.

[7] G. S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part2*, pages 115–125. Consultants Bureau, New York-London, 1962.