Symbolic Trajectory Evaluation using a SAT solver

Koen Claessen (koen@cs.chalmers.se) and Jan-Willem Roorda (jwr@cs.chalmers.se)

Introduction to STE Symbolic Trajectory Evaluation (STE) [2] is a high-performance simulation-based model checking technique. It combines three-valued simulation (using the standard values 0 and 1 together with the extra value X, "don't know") with symbolic simulation (using symbolic expressions to drive inputs). STE specifications are made in a restricted temporal language, where assertions are of the form $A \Longrightarrow C$; the antecedent A drives the simulation, and the consequent C expresses the conditions that should result. It is up to the specifier of a system to decide the balance between using expensive symbolic variables and cheap X's.

The standard implementation technique of STE is to replace the signal data type of an existing simulator with BDD based symbolic expressions resulting in 0, 1 or X. The key property of STE is that in principle only one single symbolic simulation run is needed to check whether an STE assertion holds. STE has been extremely successful in verifying properties of circuits containing large data paths (such as memories, fifos, floating point units, etc.), which are beyond the reach of traditional symbolic model checking techniques.

Disadvantages of STE The STE assertion language allows a very careful control over how many BDD variables will be introduced in the simulation process, thus controlling the size of the resulting BDDs. However, as with all BDD-based techniques, there is a possibility that BDDs will blow for a particular application.

Another disadvantage is that a simulator-based STE implementation can only propagate information forwards in time, while many properties naturally need a backwards information flow to be verified. An example of this is when the antecedent contains an assumption about an output which is needed to show something about an input mentioned in the consequent. The current formal semantics of STE only allows forwards reasoning [5]. However, there exist generalizations of STE which feature algorithms for backwards reasoning [5]. Unfortunately, these algorithms require a fixpoint computation, and are much more complicated than the original forward STE algorithm.

Our idea We would like to investigate an alternative way to implement STE, namely by replacing the simulator by a SAT solver [4]. The simulator in STE has two jobs: (1) It has to perform three-valued simulation, and (2) it has to perform symbolic simulation.

Modern SAT solvers internally *propagate* information about variables, which is linear in time and bidirectional. Moreover, the simplest form of propagation (unit resolution) is at least as powerful as three-valued simulation.

Further, when all information about variables is propagated, the SAT solver chooses a variable to branch on: the variable is independently supplied with the values 0 and 1, such that more propagation may take place. When we restrict the branching of the SAT solver to the symbolic variables used in the STE assertion, it can derive at least the same information as a symbolic simulator.

We can thus see that there is a tight match between a modern SAT solver and reasoning about STE assertions. Given an transition system $T(s, s')$, to check the STE assertion $A \Longrightarrow C$, with temporal depth n using SAT, we (roughly) create the propositional logic formula P by $(T(s_1, s_2) \wedge \ldots \wedge T(s_{n-1}, s_n) \wedge [A]) \Rightarrow [C]$. Here, $[A]$ and $[C]$ are translations of STE formulas to propositional formulas. We then try to prove P with a SAT solver which is only allowed to branch on the symbolic variables contained in A and C. Thus, we get the same complexity bound as STE via BDDs, because the complexity of the SAT procedure is exponential in the number of branches.

Advantages of performing STE via SAT One major advantage of implementing STE using SAT in the way described above is that backwards reasoning comes for free, because of the bidirectional nature of SAT-propagation. This leads to a simpler implementation of forward/backwards STE, and makes the technique easier to semantically reason about.

STE gives a weak semantics to a subset of LTL, which means that some STE properties are true in a standard LTL-semantics, but not true in STE. However, one can always transform such a property into an LTL-equivalent property that is true in STE, by introducing more symbolic variables. The user of an STE system can thus balance the line between weak reasoning power but quick verification results, and strong reasoning power but more expensive checks.

We feel that the implementation technique described here allows the user to explore this balance even more. For example, there are more powerful propagation techniques (such as *saturation* [3]) which are still polynomial but which take us beyond three-valued simulation. And what happens when we let the SAT solver branch on more variables after an STE check has come back with a counter example that contains X's?

Related Work Bjesse et al. [1] have previously used a SAT solver for STE in a different way. They simply replaced the BDDs in the simulator by propositional formulas, and used a SAT solver to check the validity of the resulting formula. Their approach also only propagates information in a forward fashion.

Results and Future Work We have implemented a translator from STE assertions to SAT problems. As an initial experiment, we have verified a memory controller (including a cache and a memory). Even without the branching restriction, the resulting SAT problems were very easy to verify!

Our next step will be to test the method on real-world examples. We hope that we can cooperate with industry in order

to find interesting test cases. We also want to investigate the effect of using stronger propagation and branching on more variables. Finally, a natural direction of future research is to investigate how to implement GSTE [5] using SAT along this line.

References

[1] P. Bjesse, T. Leonard, and A. Mokkedem. Finding bugs in an Alpha microprocessor using satisfiability solvers. In *Proceedings of the 13th International Conference of Computer-Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 454–464. Springer-Verlag, 2001.

[2] Carl-Johan H. Seger and Randal E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. *Formal Methods in System Design: An International Journal*, 6(2):147–189, March 1995.

[3] Mary Sheeran and Gunnar Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. In G. Gopalakrishnan and P. Windley, editors, *Proceedings 2nd Intl. Conf. on Formal Methods in Computer-Aided Design, FMCAD'98, Palo Alto, CA, USA, 4–6 Nov 1998*, volume 1522, pages 82–99. Springer-Verlag, Berlin, 1998.

[4] Niklas Eén & Niklas Sörensson. An extensible sat-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT2003)*, 2003.

[5] Jin Yang and C.-J. H. Seger. Introduction to generalized symbolic trajectory evaluation. In *IEEE International Conference on Computer Design: VLSI in Computers & Processors (ICCD '01)*, pages 360–367, Washington - Brussels - Tokyo, September 2001. IEEE.