Abstraction and BDDs Complement SAT-based BMC in *DiVer*

Aarti Gupta[1], Malay Ganai[1], Chao Wang[2], Zijiang Yang[1], Pranav Ashar[1]

[1] NEC Laboratories America, Princeton, NJ, U.S.A.

[2] University of Colorado, Boulder, CO, U.S.A.

Abstract. *Bounded Model Checking (BMC) based on Boolean Satisfiability (SAT) procedures has recently gained popularity for finding bugs in large designs. However, due to its incompleteness, there is a need to perform deeper searches for counterexamples, or a proof by induction where possible. The DiVer verification platform uses abstraction and BDDs to complement BMC in the quest for completeness. We demonstrate the effectiveness of our approach in practice on industrial designs.*

1 Introduction

Bounded Model Checking (BMC) [1] has recently become popular for finding bugs in large designs. In this paper, we describe the use of abstraction and BDDs to complement SAT-based BMC in our verification platform called *DiVer*, which includes backend engines for BDD-based symbolic model checking techniques [2, 3], and for Boolean Satisfiability (SAT) [4]. While BDD-based model checking provides a complete proof methodology, it works only on small models. On the other hand, SAT-based BMC can handle much larger models, but it is incomplete. Conservative abstractions, i.e. abstractions that over-approximate the paths in models, are key in providing a link between the two. In particular, we use *conservative approximations of certain state sets, computed as BDDs, as additional constraints in BMC* for various applications such as search for counterexamples [1], and proofs by induction [5]. Recent related efforts have used BDD-based enlarged targets [6], and BDD-based approximate reachability state sets [7], in searching for counterexamples with BMC.

2 Generation of BDD Constraints for BMC

Since the BDD constraints for BMC are required to be over-approximations, we obtain "existential" abstractions of the design [8], by considering some latches in the concrete design as pseudo-primary inputs. Good candidates are – latches farther in the

dependency closure of the property signals identified by localization techniques [9], peripheral/pipelining latches that do not contribute to the sequential core of the design [10, 11]. Given a conservative abstract model, and a correctness property, we use exact or approximate symbolic model checking techniques [3, 8] to generate the BDD constraints. For example, for simple safety properties, we store the union of the state sets computed iteratively by the pre-image operation, backwards from the set of bad states, as shown in Figure 1 (a). We also store the union of the state sets in the forward reachability analysis, starting from the initial state, as shown in Figure 1 (b).

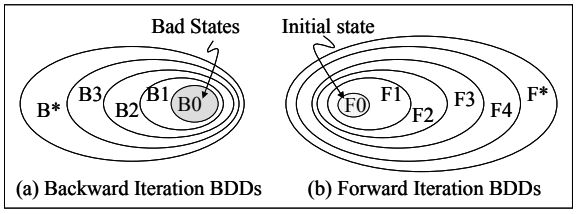

Fig. 1. Generation of BDD Constraints

We convert each BDD to either a gate-level circuit or a CNF formula, where each internal BDD node is regarded as a multiplexor controlled by that node's variable. The size of the resulting circuit or CNF formula is linear in the size of the BDD, due to introduction of extra CNF variables for each BDD node. We use reordering heuristics as well as over-approximation methods to keep down the BDD sizes.

3 Use of BDD Constraints in BMC Applications

The ability of BMC to perform a deeper search for a counterexample critically depends on the efficiency of its backend SAT solver. Modern SAT solvers [4, 12] have shown the effectiveness of adding redundant conflict clauses, which greatly improve the performance by pruning the search space, and by affecting the choice of decision variables. In a similar way, we use the BDD constraints as additional (redundant) constraints on the state variables at each (or some) cycle of the unrolled design, as shown by dark boxes in Figure 2 (a).

For safety properties, we also use the BMC engine to perform a proof by induction with increasing depth k, with restriction to loop-free paths for completeness [5]. Here, we use the BDDs not as redundant constraints, but as additional constraints in order to facilitate the proof. This is shown pictorially by the dark boxes labeled B^* and F^* in Figure 2 (b). For the base step, after unsatisfiability of the negated property at each cycle up to k cycles has been checked, we additionally check the satisfiability of the B^* BDD constraint after k cycles. If it is unsatisfiable, then the property is proved to be true. However, if B^* is satisfiable, we proceed with the inductive step, where we use the F^* BDD to constrain the arbitrary state at the start of the $k+1$ cycles. Note that this provides an additional *reachability invariant*, which can potentially allow the inductive step to succeed with BMC. It is interesting to note that the backward set B^*

complements the forward reasoning of the base step, while the forward set F^* complements the backward reasoning of the inductive step.

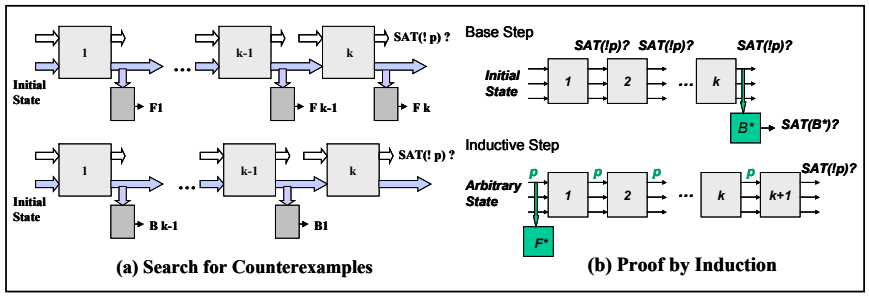

Fig. 2. Use of BDD Constraints in BMC Applications

We also use BDD constraints (forward/backward iteration BDDs) for representing sets of abstract counterexamples for simple safety properties, for performing counterexample guided abstraction refinement [13]. We are currently investigating their use for representing *witness graphs* of more general properties [14], in order to guide BMC in its search for a real counterexample, or to aid in refinement.

4 Experimental Results

We experimented with *DiVer* for checking safety properties on some large industrial designs, with more than 2k flip-flops, and more than 10k gates in the cone of influence of the property. The experiments were conducted on a Dual Intel Xeon 1700 MHz processor, with 4GB memory, running Linux 7.2, with a time limit of 3 hours. We were able to search quite deep in each design (more than 100 cycles), but could not find a counterexample. Similarly, we attempted deep proofs by induction (of depth more than 40), but the induction proofs did not succeed, despite enforcement of about 25 constraints provided by the designers.

Next, we tried BMC with BDD constraints in *DiVer*. These experiments were conducted on a 440 MHz. Sun workstation, with 1GB memory, running Solaris 5.7. The results are shown in Table 1, where Columns 2 – 5 report the results for BDD-based analysis on the abstract model, while Columns 6 – 9 report the results for a BMC-based proof by induction on the concrete design, with use of the BDD constraints. We obtained the abstract models automatically from the unconstrained designs, by abstracting away latches farther in the dependency closure of the property variables. For these experiments, we performed only the forward traversal on the abstract model, since our BDD-based symbolic model checker did not allow checking on universally constrained paths. Columns 2 – 5 report the size of the abstract model (number of flip-flops #FF, number of gates #G), the CPU time taken for traversal, the number of forward iterations, and the final size of the BDD F^*, respectively. Columns 6 – 9 report the size of the concrete design, the verification status, and the

time and memory used by the BMC engine, respectively. Note that due to the small size of the abstract models, we could keep the resource requirements for BDDs fairly low. The important point is that despite gross approximations in the abstract model analysis, the BDD reachability invariants were strong enough to let the induction proof go through successfully with BMC in each case. Though neither the BDD-based engine, nor the BMC engine, could individually prove these safety properties, their combination allowed the proof to be completed very easily (in less than a minute).

Table 1. Experimental Results for Proof by Induction using BDD-based Reachability Invariant

	BDD-based Abstract Model Analysis				Induction Proof with BDD Constraints on Concrete Design			
	#FF / #G	Time(s)	Depth	Size of F*	#FF / #G	Status	Time(s)	Mem(MB)
D1-p1	41 / 462	1.6	7	131	2198 / 14702	TRUE	0.07	2.72
D2-p2	115 / 1005	15.3	12	677	2265 / 16079	TRUE	0.11	2.84
D3-p3	63 / 1001	18.8	18	766	2204 / 16215	TRUE	0.1	2.85

References

1. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu: Symbolic Model Checking without BDDs. In: Proceedings of TACAS, vol. 1579, LNCS, 1999.
2. R.E. Bryant: Graph-based algorithms for Boolean function manipulation. In IEEE Transactions on Computers, vol. C-35(8), pp. 677-691, 1986.
3. K. L. McMillan: Symbolic Model Checking: An Approach to the State Explosion Problem. Kluwer Academic Publishers, 1993.
4. M. Ganai, L. Zhang, P. Ashar, A. Gupta, and S. Malik: Combining Strengths of Circuit-based and CNF-based SAT Algorithms for a High Performance SAT Solver. In Proceedings of Design Automation Conference, 2002.
5. M. Sheeran, S. Singh, and G. Stalmarck: Checking Safety Properties using Induction and a SAT Solver. In Proceedings of Formal Methods in Computer Aided Design, 2000.
6. J. Baumgartner, A. Kuehlmann, and J. Abraham: Property Checking via Structural Analysis. In Proceedings of Computer Aided Verification, 2002.
7. G. Cabodi, S. Nocco, and S. Quer: SAT-based Bounded Model Checking by means of BDD-based Approximate Traversals. In Proceedings of Design And Test Europe (DATE), 2003.
8. E. M. Clarke, O. Grumberg, and D. Peled: Model Checking. MIT Press, 1999.
9. R. P. Kurshan: Computer-Aided Verification of Co-ordinating Processes: The Automata-Theoretic Approach. Princeton University Press, 1994.
10. A. Gupta, P. Ashar, and S. Malik: Exploiting Retiming in a Guided Simulation Based Validation Methodology. In Proceedings of CHARME, 1999.
11. A. Kuehlmann and J. Baumgartner: Transformation-based Verification using Generalized Retiming. In Proceedings of Computer Aided Verification, 2001.
12. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik: Chaff: Engineering a Efficient SAT Solver.In Proceedings of Design Automation Conference, 2001.
13. P. Chauhan, E. M. Clarke, J. Kukula, S. Sapra, H. Veith, and D. Wang: Automated Abstraction Refinement for Model Checking Large State Spaces using SAT based Conflict Analysis. In Proceedings of Formal Methods in Computer Aided Design, 2002.
14. A. Gupta, A. E. Casavant, P. Ashar, X. G. Liu, A. Mukaiyama, K. Wakabayashi: Property-Specific Witness Graph Generation for Guided Simulation. In Proceedings of VLSI Design Conference, 2002.