A View from the Engine Room: Computational Support for Symbolic Model Checking*

Randal E. Bryant

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

1 Introduction

Symbolic model checking owes much of its success to powerful methods for reasoning about Boolean functions. The first symbolic model checkers used Ordered Binary Decision Diagrams (OBDDs) [1] to represent system transition relations and sets of system states [9]. All of the steps required for checking a model can be expressed as a series of operations on these representations, without ever enumerating individual states or transitions. More recently, bounded [3] and unbounded [10, 11] model checkers have been devised that use Boolean satisfiability (SAT) solvers as their core computational engines. Methods having a SAT solver work on a detailed system model and OBDDs operate on an abstracted model have shown that the combination of these two reasoning techniques can be more powerful than either operating on its own [4]. Boolean methods have enabled model checkers to scale to handle some of the complex verification problems arising from real-world hardware and software designs.

Given the importance of Boolean reasoning in symbolic checking, we take this opportunity to examine the capabilities of SAT solvers and BDD packages. We use several simple experimental evaluations to illustrate some strengths and weaknesses of current approaches, and suggest directions for future research.

2 Experiments in (Un)SAT

Verification problems typically use SAT solver to find an error in the design, and hence the task is to prove that a formula is unsatisfiable. Currently, the Davis-Putnam-Logemann-Loveland (DPLL) method [5] for solving SAT problems by backtracking search is heavily favored among complete SAT algorithms. Recent progress in these solvers has led to remarkable gains in speed and capacity [13], especially in proving that a formula is unsatisfiable. By contrast, using OBDDs seems like an inefficient approach to solving SAT problems, since it will generate a representation of all possible solutions, rather than a single solution. There are some common, and seemingly simple problems, for which DPLL performs poorly. We illustrate this and compare the performance of OBDDs for two sets of benchmarks.

* This research was supported by the Semiconductor Research Corporation, Contract RID 1355.001

Size	Exhaustive	LIMMAT	ZCHAFF	SIEGE	MINISAT	CUDD
8	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1
16	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1
24	3.6	10.0	0.6	0.5	0.3	< 0.1
32	TIME	TIME	13.0	3.8	3.7	< 0.1
40	TIME	TIME	TIME	72.0	162.2	< 0.1
48	TIME	TIME	TIME	TIME	TIME	< 0.1

Fig. 1. SAT Solver Performance on Parity Tree Benchmarks. Each number is the median time for comparing 16 different random trees to a linear chain. The timeout limit was set to 900 seconds.

Our first set of benchmarks compares functions for computing the parity of a set of n Boolean values using a tree of exclusive-or operators. Each instance of the problem compares a randomly generated tree to a linear chain. We generated 16 different trees for six different values of n, ranging from 8 to 48. The results, run on an 3.2 GHz Intel Pentium 4 Xeon, are shown in Figure 1. Since SAT solvers times can vary greatly depending on minor differences in the problem encoding, we report the median time for the 16 cases for each value of n. We set a timeout limit of 900 seconds for each instance.

These parity tree problems are known to be difficult cases for DPLL, or in fact any method based on resolution principle. Zhang and Malik submitted one such problem for $n = 36$ as a benchmark for the SAT solver competition held in conjunction with the SAT 2002 [14]. None of the solvers at the time could solve the problem within the 40-minute time limit on an Intel Pentium III.

We tested six different solution methods:

Exhaustive Enumerate all 2^n possible solutions and test each one.

LIMMAT The LIMMAT solver by Armin Biere from 2002. This solver uses the innovations introduced by the GRASP [8] and CHAFF solvers [13], but without the level of tuning found in more recent solvers.

ZCHAFF The ZCHAFF 2004 solver, carrying on the work by Malik and his students [13].

SIEGE The SIEGE version 4 solver, winner of the 2003 SAT competition.

MINISAT The MINISAT version 1.14 solver, winner of the 2005 SAT competition.

CUDD An OBDD-based implementation using the CUDD library from the University of Colorado.

As can be seen from the results of Figure 1, exhaustive evaluation can readily handle problems up to $n = 24$, but it becomes impractical very quickly due to the exponential scaling. The LIMMAT solver actually performs slightly worse than exhaustive evaluation. The other DPLL solvers (ZCHAFF, SIEGE, and MINISAT) can handle all 16 instances for $n = 32$. Only MINISAT can handle all 16 instances for $n = 40$, and even it can solve only 4 of the 16 instances for $n = 48$. These experiments illustrate that DPLL solvers have progressed considerably since the 2002 SAT competition, but they all experience exponential growth for this class of problems.

By contrast, OBDDs can solve parity tree problems with hardly any effort, never requiring more than 0.1 seconds for any of the instances. Parity functions have OBDD representations of linear complexity [1], and hence the tree comparison problem can be solved in worst-case $O(n^2)$ time using OBDDs. It would be feasible to solve instances of this problem for $n = 1000$ or more.

As a second class of benchmarks, we consider ones that model the bit-level behavior of arithmetic operations. Consider the problem of proving that integer addition and multiplication are associative. That is, we wish to show that the following two C functions always return the value 1:

```
int assocA(int x, int y, int z)
{
  return (x+y)+z = x+(y+z);
}

int assocM(int x, int y, int z)
{
  return (x*y)*z = x*(y*z);
}
```

We created a set of benchmark problems from these C functions, where we varied the number of bits n in the argument words `x`, `y`, and `z`, up to a maximum of $n = 32$. Since there are three arguments, the number of possible argument combinations is 8^n.

Problem	Exhaustive	CUDD	MINISAT
Addition	12	> 32	> 32
Multiplication	12	8	5

Fig. 2. Performance in Solving Associativity Problems Numbers indicate the maximum word size that can be solved in under 900 seconds of CPU time.

Figure 2 shows the performance for this benchmark by exhaustive evaluation, OBDDs using the CUDD package, and MINISAT (the best DPLL-based approach tested). In each case, we show the maximum number of argument bits n for which the problem can be solved within a 900 second time limit. Exhaustive evaluation works up to $n = 12$, but then becomes impractical, with each additional bit requiring eight times more evaluations. Both DPLL and OBDDs can show that addition is associative up to the maximum value tested. For multiplication, we see that OBDDs outperform DPLL, but neither does as well as brute-force, exhaustive evaluation. For OBDDs, we know that the Boolean functions for integer multiplication require OBDDs of exponential size [2], and hence OBDD-based methods for this problem incur exponential space and time. Evidently, DPLL-based methods also suffer from exponential time performance.

3 Observations

Our first set of benchmarks illustrates one of the challenges of the Boolean satisfiability problem. While DPLL works well on many problems, it has severe weaknesses, some of which can be filled by more "niche" approaches, such as OBDDs. Some attempts have been made to develop SAT solvers that use different combinations of DPLL and OBDDs, e.g., [7], but none of these has demonstrated consistent improvements over DPLL. In particular, it seems like the main advantage of current DPLL/OBDD hybrids is that they can solve problems that are tractable using either DPLL or OBDDs. We have not seen meaningful examples of them solving problems that cannot be solved by one of the two approaches operating in isolation.

An additional concern of ours is that the recent success of DPLL methods is having the effect that the research field is narrowly focusing on this approach to SAT solving. Researchers have found they can do better in SAT competitions by fine tuning DPLL solvers rather than trying fundamentally different approaches. While this tuning has led to remarkable improvements, it is not healthy for the field to narrow the "gene pool" of SAT techniques. Rather, we should be encouraging researchers to explore new approaches, even if they only outperform DPLL on small classes of problems, as long as these classes have practical applications. Steps in this direction include recent work by Jain, et al [6].

Our arithmetic problems illustrate that, while both DPLL and OBDDs are adequate for addition and related functions, neither performs well for operations related to integer multiplication. Indeed, companies that market circuit equivalence checkers have had to devise *ad hoc* workarounds for checking circuits containing multipliers. We believe that the rresearch community should invest more effort in tackling problems that are well beyond the capability of existing SAT solvers. Examples of challenging problems arise in the fields of cryptanalysis [12] and combinatorial optimization.

4 Conclusion

2006 marks the twenty-fifth anniversary of model checking, but also the twentieth anniversary of powerful tools for Boolean reasoning, first with OBDDs and more recently with DPLL-based SAT solvers. The field has advanced considerably due to both clever ideas and careful engineering. Model checking and many other application areas have directly benefited from these tools. It is important that the research community keeps pushing ahead with new approaches and new improvements in Boolean reasoning. There remain many important problems that are beyond the reach of today's methods.

References

1. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
2. R. E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2):205–213, February 1991.

3. 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.
4. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *JACM*, 50(5):752–794, 2003.
5. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communcations of the ACM*, 5(7):394–397, 1962.
6. H. Jain, C. Bartzis, and E. M. Clarke. Satisfiability checking of non-clausal formulas using general matings. In A. Biere and C. P. Gomes, editors, *Proceedings of Ninth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2006)*, LNCS 4121, pages 75–89, 2006.
7. H. Jin and F. Somenzi. CirCus, a hybrid satisfiability solver. In H. H. Hoos and D. G. Mitchell, editors, *Proceedings of Eighth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2005)*, LNCS 3542, pages 211–223, 2005.
8. J. P. Marques-Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.
9. K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1992.
10. K. McMillan. Applying SAT methods in unbounded symbolic model checking. In E. Brinksma and K. G. Larsen, editors, *Computer-Aided Verification (CAV '02)*, LNCS 2404, pages 250–264, 2002.
11. K. McMillan. Interpolation and SAT-based model checking. In W. A. Hunt, Jr. and F. Somenzi, editors, *Computer-Aided Verification (CAV '03)*, LNCS 2725, pages 1–13, 2003.
12. I. Mironov and L. Zhang. Applications of SAT solvers to cryptanalysis of hash functions. In A. Biere and C. P. Gomes, editors, *Proceedings of Ninth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2006)*, LNCS 4121, pages 102–115, 2006.
13. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *38th Design Automation Conference (DAC '01)*, pages 530–535, 2001.
14. L. Simon, D. Le Berre, and E. A. Hirsch. The SAT2002 competition. *Annals of Mathematics and Artificial Intelligence*, 43(1–4), 2005.