SATIRE: A New Incremental Satisfiability Engine

Jesse Whittemore
University of Michigan
jwhitte@umich.edu

Joonyoung Kim
Intel Corporation
joonyoung.kim@intel.com

Karem Sakallah
University of Michigan
karem@umich.edu

ABSTRACT

We introduce SATIRE, a new satisfiability solver that is particularly suited to verification and optimization problems in electronic design automation. SATIRE builds on the most recent advances in satisfiability research, and includes two new features to achieve even higher performance: a facility for incrementally solving sets of related problems, and the ability to handle non-CNF constraints. We provide experimental evidence showing the effectiveness of these additions to classical satisfiability solvers.

1. INTRODUCTION

Recent advances in backtrack search algorithms for Boolean satisfiability (SAT) have enabled the utilization of SAT modeling in a wide variety of large-scale EDA applications such as combinational and sequential equivalence checking, bounded model checking, test pattern generation, timing analysis, and FPGA routing. In particular, algorithmic enhancements to basic backtrack search such as conflict analysis [17] and recursive learning [12], coupled with optimized software implementations [20], make it possible today to solve much larger SAT instances (thousands of variables and millions of clauses) than what was possible just a few years ago. Building on this success, we introduce two enhancements that expand the modeling range of SAT in ways that directly correspond to characteristics typical of problems from the EDA field. Specifically, we address applications (1) that require the solution of many related SAT instances [10][11] and (2) that involve complex constraints whose encoding in conjunctive normal form (CNF) is impractical. These enhancements have been incorporated in SATIRE, a new backtrack SAT program that uses an incremental reasoning engine to efficiently solve sets of related SAT instances. In addition to CNF, SATIRE also accepts a variety of constraint types including pseudo-Boolean inequalities [1] and cube lists. SATIRE employs conflict analysis to derive and record conflict clauses and is architected to allow the recording of clauses identified through hypothetic reasoning on both variables (recursive learning) and clauses [15][18].

The rest of this paper is organized in 4 sections. In Section 2, we define an extension of classical SAT supporting the cooperative solution of a set of related SAT instances as well as accommodating non-CNF constraints. Section 3 addresses some of the implementation issues that must be solved to incorporate these extensions in a modern SAT solver. The effectiveness of these enhancements to classical SAT is demonstrated by a sampling of experimental results in Section 4. Conclusions and future directions are summarized in Section 5.

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

2. PROBLEM STATEMENT

The extended satisfiability problem we address is that of solving a series of k related SAT instances $\{\varphi_1, \ldots, \varphi_k\}$ defined over n binary variables $x_1, \ldots, x_n$. Each φ_i is a set of constraints that must be simultaneously satisfied, and the consecutive instances are related according to:

$$\varphi_{i+1} = (\varphi_i - \rho_i) \cup \alpha_{i+1}$$

where ρ_i and α_{i+1} represent sets of constraints that are, respectively, removed/added to transform φ_i into φ_{i+1}. The constraints can be expressed in a variety of formats including:

- Standard CNF, e.g. $(x_3 \vee \neg x_7 \vee x_{12})$
- Sum of products, e.g. $(x_2 \neg x_5 \vee \neg x_3 \neg x_4 x_{11} \vee x_6 x_9)$
- Pseudo-Boolean inequalities, e.g. $(3x_1 - \neg x_2 + 2x_3 \geq 2)$

Solving $\{\varphi_1, \ldots, \varphi_k\}$ incrementally is accomplished by starting the search for a solution to φ_{i+1} from the solution found for φ_i and by taking advantage of all the learning (through clause recording) that occurred during the solution of $\{\varphi_1, \ldots, \varphi_i\}$. The effectiveness of incremental search, thus, depends on the degree of similarity between the instances. In particular, one should expect the maximum benefit to accrue when $|\rho_i|, |\alpha_{i+1}| \ll |\varphi_{i+1}|$. In addition, for some sets of SAT instances significant computational savings can result from "early termination", namely when the unsatisfiability of φ_i implies the unsatisfiability of all subsequent instances and obviates the need to solve them. It is interesting to note that this template generalizes earlier efforts at incremental satisfiability [2][9].

EDA applications that lend themselves well to this extended SAT formulation include (1) path sensitization problems such as test pattern generation and timing analysis and (2) state traversal problems such as sequential equivalence and model checking. Path sensitization applications typically require the solution of a set of related problems (find tests for all stuck-at faults; find input patterns to propagate events along the circuit's longest paths) that differ only slightly from each other. For example, the SAT instance for a particular stuck-at fault problem consists of a large set of common con-

straints that model the logic circuit, and a much smaller set of specific constraints that model fault activation and propagation.

State traversal problems, in contrast, involve unrolling a sequential circuit for a number of cycles. Assuming that r_i models the state transition constraints from cycle $i-1$ to cycle i, the SAT instance φ_{i+1} can be expressed as $[r_1 \cup \ldots \cup r_{i+1}] \cup \alpha_{i+1}$ where α_{i+1} represents the additional constraints that must be satisfied in cycle $i+1$ (typically translated from a temporal logic specification.) The structure of these problems admits an incremental solution approach that can take advantage of early termination as well as learning across cycles of operation.

In addition to the above EDA applications, combinatorial optimization problems with pseudo-Boolean objective functions can be solved as sequences of related SAT instances that differ only in a single pseudo-Boolean constraint which becomes tighter as the search for the optimal solution progresses [1]. Such optimization-to-decision problem transformations are well known, but take on added value in the context of incremental satisfiability because of the potential of learning across iterations of the optimization process. Optimization applications also highlight the need for constraints that are more expressive than CNF. Specifically, minimizing an objective function of the form $\sum a_i x_i$, where the a_i coefficients are real and the optimization variables x_i are binary, leads to pseudo-Boolean constraints $\sum a_i x_i \leq b_j$ whose CNF representation can be exponentially large.

3. IMPLEMENTATION ISSUES

SATIRE is an implementation of a solver for problems of the general SAT form described above. It has its roots in the GRASP [16] package, building upon its conflict diagnosis technology. In addition, SATIRE was architected to allow for the integration of additional forms of Boolean reasoning such as Recursive Learning (RL) [12] and Stalmarck's Method (SM) [8][18], as well as the inclusion of arbitrary constraint types and incremental satisfiability.

Conflict Diagnosis. SATIRE depends on the technique known as *clause recording*, which adds clauses to the original problem set based on the analysis of conflicts encountered during the search. SATIRE's implementation is adapted from GRASP with the notable difference that relationships between learned clauses and existing constraints must be determined during conflict analysis. If constraints are removed from a problem (via incremental satisfiability) it is necessary to also remove any learned clauses that were derived from those constraints. The interdependency between newly created clauses and their previously existing "parent" constraints gives rise to a *constraint hierarchy* that may grow many levels deep.

Constraint Modeling. When applied to pure CNF instances, SATIRE's (as well as GRASP's) reasoning engines rely on the unit clause rule [7] to generate implications, diagnose conflicts, and perform nonchronological backtracking. The CNF form is central to these tasks because conflict signatures are inadmissible variable assignments whose negation exactly corresponds to the clausal form.

A natural question when considering non-CNF constraints is whether the unit clause rule needs to be extended to a "unit constraint rule." For example, the pseudo-Boolean constraint $(a + b + c + d + e \geq 4)$ becomes *unit* under the assignment $a = 0$ yielding the implication $\neg a \rightarrow bcde$, forcing the remaining unassigned variables to become 1. It turns out that implementing such a rule for non-CNF constraints is unnecessary for completeness of the search but can have a dramatic impact on its efficiency. In particular, failure to imply necessary assignments from a non-CNF constraint causes SATIRE's conflict diagnosis procedure to create corresponding CNF clauses that, individually, partially cover the original non-CNF constraint. Over time, a complete CNF equivalent representation of the non-CNF constraint may be accumulated. This can be viewed as a feature for constraint types where an efficient implementation of the unit constraint rule is difficult or computationally expensive; SATIRE will automatically perform an "on-demand" transformation to CNF creating only those clauses that are necessary for backtracking. In practice, however, the unit constraint rule greatly improves the efficiency of the solver and should be implemented whenever possible. In the current implementation of SATIRE, the unit rule is fully supported for pseudo-Boolean inequalities and partially supported for sum-of-product constraints.

4. EXPERIMENTAL VALIDATION

In this section we present examples of SATIRE's special features being applied to the problems of functional delay fault testing [14][19] and bounded model checking [3][4]. These experiments demonstrate the natural match between incremental satisfiability with extended constraint types and EDA problem domains. Both examples are modeled by SATIRE in straightforward manners, and in each case SATIRE's capabilities lead to improved performance.

Minimal FDFT. Ideally, one would test all paths through a circuit for delay faults. The potentially exponential number of paths makes such an approach infeasible, however, and functional delay fault testing (FDFT) has emerged as a compromise [14]. Under this schema, we would seek two-vector test patterns that cause the propagation of a transition t_I from an input I to an output O where it appears as transition t_O, but the actual path is left unspecified.

We extended the basic FDFT by requiring *minimal* test patterns; test patterns with the fewest possible specified inputs (i.e., the most don't cares). Some FDFT methodologies require tens to hundreds of tests per input/output/transition combination to ensure adequate coverage [14]. Thus, given a choice, a test pattern with more unspecified inputs is preferable to one with fewer since the former can be used to generate more tests [19]. Such an objective function is readily cast as a pseudo-Boolean inequality $x_1 + \ldots + x_n \leq b$ using a suitable 2-bit encoding of the problem variables.

Minimal FDFT pattern generation requires the solution of an optimization problem for each input/output/transition combination of the circuit under test. Each subproblem, encoded as SAT, is comprised of three groups of constraints: (1) a large set of CNF clauses modeling the circuit's logical behavior, (2) a few trivial constraints to force the desired input/output transitions, and (3) objective con-

straints that limit the number of specified inputs. As an optimization problem, the SAT procedure will be iterated with the objective constraints being made more restrictive until the minimum number of specified inputs is found.

The objective constraint is of particular interest because it can be implemented by either CNF clauses or a pseudo-Boolean inequality. We performed minimal FDFT pattern generation for the combinational portions of a subset of the ISCAS89 circuits using both methods. For each benchmark circuit, $2 \times |I| \times |O|$ optimization instances (corresponding to both transition combinations for all possible input/output pairs) were created and solved. The results of this experiment are shown in Table 1: column 1 gives the circuit name, and column 2 shows the number of primary inputs.[1] The range in the optimal number of specified inputs for each circuit is given in columns 3 and 4. For example, across all optimization instances for circuit s208.1 with 18 inputs, the number of specified inputs in the optimal tests ranged between 2 and 12. The last three columns of the table show the search times to find these optimal solutions using three different configurations of SATIRE. The data in columns 5 and 6 were generated without invoking the incremental reasoning engine. In other words, the sequence of SAT instances for each optimization iteration as well as the separate SAT instances for each input/output/transition combination were solved in isolation. The data in column 7 were produced by turning on the incremental reasoning engine within individual, as well as across, optimization instances.

TABLE 1: Minimal FDFT Results

Ckt	#Inputs	# Specified inputs in solution		Solution time (sec)		
				CNF	CNF + PB	
		Min	Max	NonInc		Inc
s208.1	18	2	12	1159	7	3
s344	24	1	11	-	52	33
s349	24	1	11	-	52	35
s382	24	1	12	-	34	17
s386	13	4	11	63	13	6
s400	24	1	12	-	37	21
s420	35	2	20	-	42	17
s444	24	1	12	-	43	22
s510	25	3	8	-	62	21
s526	24	1	14	-	49	29
s820	23	3	13	-	129	59
s832	23	3	13	-	127	58
s1196	32	1	18	-	2039	1701
s1238	32	1	18	-	2027	1701
s1488	14	3	11	857	413	234
s1494	14	3	11	863	410	234

[1] Technically, this corresponds to the number of primary inputs and flip-flops in the original sequential circuits.

The column labeled "CNF" indicates the run times when the pseudo-Boolean objective function is modeled by a set of CNF clauses during the search process. The data in this column clearly suggest the infeasibility of such an approach as only four out of the sixteen circuits could be processed this way. For the remaining twelve benchmarks, in excess of 600,000 objective clauses were generated before exhausting the 500MB of available memory. The columns labeled "CNF+PB" show the data generated when the circuit and transition constraints were modeled by CNF clauses, whereas the objective functions were cast as pseudo-Boolean inequalities. The run times in these two columns contrast the cases of non-incremental and fully incremental search. In both cases, all optimization runs were completed yielding identical optimal solutions and were significantly faster than the successful CNF-only runs. This should be hardly surprising since the CNF-only cases were significantly larger in size. The incremental approach was also consistently faster, yielding an average speedup of 2 over the non-incremental approach.

For comparison, we ran the same tests using the binate covering solver *scherzo* [5][6]. Scherzo was moderately successful when it was not instructed to minimize the number of specified inputs. It was not competitive, however, when it attempted to generate *minimal* FDFT patterns, with performance ranging from 1.5 to several hundred times slower than the SATIRE results.

Incremental BMC. Bounded model checking (BMC) [4][3] is the process of verifying that a property holds in a transition system for all paths up to a specified length. Length bounds are commonly tested in succession because higher bounds are typically much harder to verify, and thus a significant amount of work can be avoided if a failing property can be detected with smaller bounds. Additionally, shorter (simpler) counter examples are desirable for failing properties.

A SAT formulation of BMC includes a set of constraints for each time step i modeling the transition from the preceding step $i-1$. A final set of constraints specifies the property being tested. In an incremental framework, the individual time step and property check constraints for each corresponding length bound are added incrementally, testing each bound along the way. This allows the model checker to identify the shortest counter example in the case of a failing property without starting from scratch for each bound.

Table 2 contains the times, in seconds, required by SATIRE to verify 10 properties of the PCI Local Bus [13][16], both incrementally and non-incrementally, for paths of length 1 to 14. The values in the "Non-Inc" and "Inc" columns compare the cumulative times required to test each length individually versus the run time of the same solver incrementally solving for all 14 bounds. The incremental verifications are consistently faster, sometimes dramatically so.

5. CONCLUSIONS AND FUTURE RESEARCH

The performance and capacity of satisfiability solvers has been increasing steadily over the past few years. This has accelerated their adoption as the preferred Boolean reasoning engine in many EDA

TABLE 2: Incremental BMC of the PCI Bus[1]

#	Property	Non-Inc	Inc
1	Transaction Termination	2.4	1.5
2	Bus Arbitration	118.2	64.9
3	Target Termination	19.7	6.4
4	Target Termination	3.0	1.8
5	Read/Write Transaction	19.6	3.9
6	Data Transfer	375.4	318.3
7	Device Selection	1584.6	701.6
8	Device Selection	7.6	5.3
9	Master Abortion	1533.3	663.2
10	Master Abortion	7.2	5.44

[1] The properties were obtained from [16]; we translated them into CNF for these experiments.

applications, especially in situations where symbolic methods based on BDDs had limited success. This paper introduced two new enhancements to the classical SAT formulation that were motivated by the particular properties of EDA problems. The first enhancement allows sets of related problems—a common occurrence in both verification and optimization scenarios—to be solved incrementally. Compared to solving such related problems separately, the incremental approach achieves a significant performance advantage. The second enhancement—the ability to handle more expressive constraint forms—extends the range of problems that can be attacked beyond what is feasible with classical CNF SAT. In particular, pseudo-Boolean inequalities which are a more natural form than CNF for representing objective functions in optimization problems, allow SAT solvers to become very competitive contenders for solving combinatorial optimization problems.

SATIRE was architected to facilitate the incorporation of many of the features proposed in recent years to enhance the effectiveness of the search process. We plan to use SATIRE as an experimental testbed for the evaluation of such enhancements as well as the development of additional ones. Our long-term goal is to explore the limits of applicability of SAT methods in EDA.

6. ACKNOWLEDGMENTS

This work was funded in part by a grant from the Intel Corporation. The authors would also like to acknowledge Fadi Aloul for providing the incremental BMC data.

7. REFERENCES

[1] P. Barth, "A Davis-Putnam based Enumeration Algorithm for Linear Pseudo-Boolean Optimization," Technical Report MPI-I-95-2-003, Max-Planck-Institut Für Informatik, 1995.

[2] H. Bennaceur, I. Gouachi and G. Plateau, "An Incremental Branch-and-Bound Method for the Satisfiability Problem," *INFORMS Journal on Computing*, vol. 10, pp. 301-308, 1998.

[3] A. Biere, A. Cimmati, E. Clarke, M. Fujita, and Y. Zhu, "Symbolic Model Checking using SAT Procedures instead of BDDs," *Design Automation Conference (DAC)*, pp. 317-320, 1999.

[4] E. Clarke, E. Emerson, and A. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic," *ACM Transactions on Programming Languages and Systems*, pp. 244-263, 1986.

[5] O. Coudert and J. C. Madre, "New Ideas for Solving Covering Problems," *Design Automation Conference (DAC)*, pp. 641-646, 1995.

[6] O. Coudert, "On Solving Covering Problems," *Design Automation Conference (DAC)*, pp. 197-202, 1996.

[7] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem Proving," *Communications of the ACM*, vol. 5, pp. 394-397, 1962.

[8] J. F. Groote and J. P. Warners, "The Propositional Formula Checker HeerHugo," Technical Report SEN-R9905 (CWI), 1999.

[9] J. N. Hooker, "Solving the Incremental Satisfiability Problem," *Journal of Logic Programming*, vol. 15, pp. 177-186, 1993.

[10] J. Kim, J. Whittemore, J. P. M. Silva and K. A. Sakallah, "On Application of Incremental Satisfiability to Delay Fault Testing," *Design, Automation and Test in Europe (DATE)*, pp. 380-384, 2000.

[11] J. Kim, J. Whittemore, J. P. M. Silva and K. A. Sakallah, "On Solving Stack-Based Incremental Satisfiability Problems," *International Conference on Computer Design (ICCD)*, pp. 379-382, 2000.

[12] W. Kunz and D. Stoffel, *Reasoning in Boolean Networks*, Kluwer Academic Publishers, 1997.

[13] PCI Special Interest Group, "PCI Local Bus Specification," Revision 2.2, December 1995.

[14] I. Pomeranz, S. M. Reddy, "Functional Delay Faults in Macro-based Combinational Circuits," *International Conference on Computer-Aided Design (ICCAD)*, pp. 687-694, 1995.

[15] M. Sheeran and G. Stalmarck, "A Tutorial on Stalmarck's Proof Procedure for Propositional Logic," *Proceedings of Formal Methods in Computer-Aided Design (FMCAD)*, pp. 82-99, 1998.

[16] K. Shimizu, D. Dill, and A. Hu, "Monitor-Based Formal Specification of PCI," *Proceedings of Formal Methods in Computer-Aided Design (FMCAD)*, pp. 335-353, 2000.

[17] J. P. Marques-Silva, and K. A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506-521, May 1999.

[18] G. Stalmarck, "A System for determining propositional logic theorems by applying values and rules to triplets that are generated from boolean formula," Swedish Patent No. 467,076 (approved 1992), U.S. Patent No. 5,276,897 (approved 1994), European Patent No. 0403 454 (approved 1995).

[19] S. Tragoudas and M. Michael, "ATPG tools for Delay Faults at the Functional Level," *Design, Automation and Test in Europe (DATE)*, pp. 631-635, 1999.

[20] H. Zhang, "SATO: An Efficient Propositional Prover," in *Proceedings of International Conference on Automated Deductio*n, pp. 272-275, July 1997.