Boolean Satisfiability in Electronic Design Automation

João P. Marques-Silva
Departments of Informatics
Technical University of Lisbon, IST/INESC
Cadence European Labs
Lisbon, Portugal
jpms@inesc.pt

Karem A. Sakallah
Department of EECS
University of Michigan
Ann Arbor, Michigan
karem@eecs.umich.edu

Abstract

Boolean Satisfiability (SAT) is often used as the underlying model for a significant and increasing number of applications in Electronic Design Automation (EDA) as well as in many other fields of Computer Science and Engineering. In recent years, new and efficient algorithms for SAT have been developed, allowing much larger problem instances to be solved. SAT "packages" are currently expected to have an impact on EDA applications similar to that of BDD packages since their introduction more than a decade ago. This tutorial paper is aimed at introducing the EDA professional to the Boolean satisfiability problem. Specifically, we highlight the use of SAT models to formulate a number of EDA problems in such diverse areas as test pattern generation, circuit delay computation, logic optimization, combinational equivalence checking, bounded model checking and functional test vector generation, among others. In addition, we provide an overview of the algorithmic techniques commonly used for solving SAT, including those that have seen widespread use in specific EDA applications. We categorize these algorithmic techniques, indicating which have been shown to be best suited for which tasks.

1 Introduction

Recent years have seen a tremendous growth in the number of R&D groups at Electronic Design Automation (EDA) companies, universities and research laboratories, that have started using Boolean Satisfiability (SAT) models and algorithms for solving different problems in EDA. Despite SAT being an NP-complete decision problem, SAT algorithms have seen dramatic improvements in recent years, allowing larger problem instances to be solved in different application domains [4, 24, 27, 42]. Moreover, dedicated SAT algorithms that target solving instances from EDA problems have been proposed [26, 37]. These algorithms exploit the specific structure of most instances from EDA problems and incorporate techniques for solving SAT problems in digital circuits. It is reasonable to expect further improvements to SAT algorithms as more attention is focused on the practical application of SAT to real design problems in EDA.

The main purpose of this paper is to review existing applications of SAT to EDA, and to survey modern SAT algorithms, emphasizing solutions that have been shown effective for EDA applications.

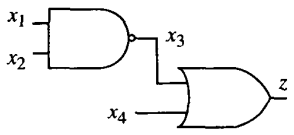
The paper is organized as follows. Section 2 introduces the definitions used throughout the paper, emphasizing Conjunctive Normal Form (CNF) formulas and the representation of circuits in CNF. Afterwards, we review algorithms for Boolean Satisfiability, giving emphasis to the algorithms that are recognizably more suitable for EDA applications, and address the extension of the well-known recursive learning paradigm to CNF formulas. Section 5 addresses specific solutions for solving SAT in combinational circuits. Section 6 reviews recent work that has shown promise for solving SAT in EDA applications. Section 7 concludes the paper.

2 Definitions

A conjunctive normal form (CNF) formula φ on n binary variables $x_1, \ldots, x_n$ is the conjunction of m *clauses* $\omega_1, \ldots, \omega_m$ each of which is the disjunction of one or more literals, where a literal is the occurrence of a variable x or its complement x'. A formula φ denotes a unique n-variable Boolean function $f(x_1, \ldots, x_n)$ and each of its clauses corresponds to an implicate of f. Clearly, a function f can be represented by many equivalent CNF formulas. The satisfiability problem (SAT) is concerned with finding an assignment to the arguments of $f(x_1, \ldots, x_n)$ that makes the function equal to 1 or proving that the function is equal to the constant 0.

The CNF formula of a combinational circuit is the conjunction of the CNF formulas for each gate output, where the CNF formula of each gate denotes the valid input-output assignments to the gate. An example of a circuit, associated CNF formula and the specification of an objective is shown in Figure 1. (The derivation of the CNF formulas for simple gates is shown in Table 1 [20].) If we

DAC 2000, Los Angeles, California



$$\varphi = (x_1 + x_3) \cdot (x_2 + x_3) \cdot (\neg x_1 + \neg x_2 + \neg x_3) \cdot (\neg x_3 + z) \cdot (\neg x_4 + z) \cdot (x_3 + x_4 + \neg z)$$

(a) Consistent assignments

$$\varphi' = (x_1 + x_3) \cdot (x_2 + x_3) \cdot (\neg x_1 + \neg x_2 + \neg x_3) \cdot (\neg x_3 + z) \cdot (\neg x_4 + z) \cdot (x_3 + x_4 + \neg z) \cdot (\neg z)$$

(b) With property $z = 0$

Figure 1: Example circuit and CNF formula

Gate type	Gate function	φ_x
AND	$x = \text{AND}(w_1, \ldots, w_j)$	$\left[\prod_{i=1}^{j} (w_i + \neg x)\right] \cdot \left(\sum_{i=1}^{j} \neg w_i + x\right)$
NAND	$x = \text{NAND}(w_1, \ldots, w_j)$	$\left[\prod_{i=1}^{j} (w_i + x)\right] \cdot \left(\sum_{i=1}^{j} \neg w_i + \neg x\right)$
OR	$x = \text{OR}(w_1, \ldots, w_j)$	$\left[\prod_{i=1}^{j} (\neg w_i + x)\right] \cdot \left(\sum_{i=1}^{j} w_i + \neg x\right)$
NOR	$x = \text{NOR}(w_1, \ldots, w_j)$	$\left[\prod_{i=1}^{j} (\neg w_i + \neg x)\right] \cdot \left(\sum_{i=1}^{j} w_i + x\right)$
NOT	$x = \text{NOT}(w_1)$	$(x + w_1) \cdot (\neg x + \neg w_1)$
BUFFER	$x = \text{BUFFER}(w_1)$	$(\neg x + w_1) \cdot (x + \neg w_1)$

Table 1: CNF formulas for simple gates

view a CNF formula for a gate as a set of clauses, the CNF formula φ for the circuit is defined by the set union (or conjunction) of the CNF formulas of each gate. Hence, given a combinational circuit it is straightforward to create the CNF formula for the circuit as well as the CNF for proving a given property of the circuit.

SAT algorithms operate on CNF formulas, and consequently can readily be applied to solving instances of SAT associated with combinational circuits.

3 SAT Applications in EDA

This section briefly surveys the application of SAT models to EDA applications. (See [31] for a more detailed account of some of the earlier applications.) One of the most well-know applications is Automatic Test Pattern Generation (ATPG) [20, 25, 38]. Other applications in

```
// Input arg:       Current decision level d
// Output arg:      Backtrack decision level β
// Return value:    SATISFIABLE or UNSATISFIABLE
//
SAT (d, &β)
{
    if (Decide (d) != DECISION)
        return SATISFIABLE;
    while (TRUE) {
        if (Deduce (d) != CONFLICT) {
            if (SAT (d + 1, β) == SATISFIABLE)
                return SATISFIABLE;
            else if (β != d || d == 0) {
                Erase (d); return UNSATISFIABLE;
            }
        }
        if (Diagnose (d, β) == CONFLICT) {
            return UNSATISFIABLE;
        }
    }
}
```

Figure 2: Generic backtrack search SAT algorithm

testing include delay fault testing [7] and redundancy identification and elimination [17].

Besides testing, SAT models have been used in circuit delay computation [28, 36], FPGA routing [29, 30], logic synthesis [12] and, recently, in crosstalk noise analysis [8]. SAT models have also been used for functional vector generation [13].

With respect to circuit verification, SAT models have found several applications. Combinational equivalence checking can easily be cast as an instance of SAT, and different approaches have been proposed [16, 19, 26]. Additional work has included processor verification [6] and bounded model checking [5].

SAT can also be used for solving linear integer optimization problems [3], with immediate potential applications in solving covering problems [9], in computing prime implicants of Boolean functions [22] and in physical design problems [35]. An example of a SAT-based covering algorithm is described in [23].

4 Algorithms for Satisfiability

Over the years several approaches have been proposed for solving SAT, including local search [32], backtrack search [11], continuous formulations [33] and algebraic manipulation [15, 34]. Of these, only backtrack search has proven useful for solving instances of SAT from EDA applications, in particular for applications where the objective is to prove unsatisfiability. In this section we review modern backtrack search algorithms for SAT and describe recent extensions of the recursive learning paradigm [19] to solving SAT.

4.1 Backtrack Search

The overall organization of a generic backtrack search SAT algorithm is shown in Figure 2. This generic SAT algorithm captures the organization of several of the most competitive algorithms [4, 27, 42]. The algorithm con-

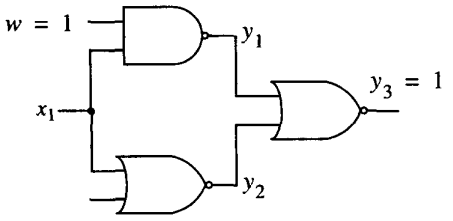


Figure 3: Example circuit

ducts a search through the space of the possible assignments to the problem instance variables. At each stage of the search, a variable assignment is selected with the `Decide()` function. A decision level d is associated with each selection of an assignment. Implied necessary assignments are identified with the `Deduce()` function, which in most cases corresponds to straightforward derivation of implications [10, 41]. Whenever a clause becomes unsatisfied the `Deduce()` function returns a conflict indication which is then analyzed using the `Diagnose()` function. The diagnosis of a given conflict returns a backtracking decision level β, which denotes the decision level to which the search process is required to backtrack to. The `Erase()` function clears implied assignments that result from each assignment selection. Different organizations of SAT algorithms can be modeled by this generic algorithm. Currently, all of the most efficient SAT algorithms [4, 27, 42] are characterized by several of the following key properties:

1. The analysis of conflicts can be used for implementing *Non-chronological Backtracking* search strategies. Hence, assignment selections deemed irrelevant can be skipped over during the search [4, 27, 42].
2. The analysis of conflicts can also be used for identifying and recording new implicates of the Boolean function associated with the CNF formula. *Clause Recording* plays a key role in recent SAT algorithms, but in most cases large recorded clauses are eventually deleted [4, 27, 42].
3. Other techniques have been developed. *Relevance-Based Learning* [4] extends the life-span of large recorded clauses that will eventually be deleted. *Conflict-Induced Necessary Assignments* [27] denote assignments to variables which are necessary for preventing a given conflict from occurring again during the search.

Before running the SAT algorithm, different forms of preprocessing can be applied [27]. This in general is denoted by a `Preprocess()` function.

The techniques that characterize modern backtrack search SAT algorithms are based on the ability to analyze the causes of conflicts during the search and deriving *explanations* for those conflicts. For example, let us consider the example circuit of Figure 3, where $w = 1$ and $y_3 = 0$, and x_1 is assigned value 1. Clearly, this assignment yields a conflict, since y_1 and y_2 are both assigned

Assignments: $\{z = 1, u = 0\}$

$$\omega_1 = (u + x + \neg w)$$
$$\omega_2 = (x + \neg y)$$
$$\omega_3 = (w + y + \neg z)$$

Figure 4: Recursive learning on clauses

value 0, and these assignments are inconsistent with the assignment of node y_3. This conflict will hold as long as the assignments $x_1 = 1$, $w = 1$ and $y_3 = 0$ hold. Hence, in order to prevent this conflict at least one of the assignments must be complemented. As a result, the clause $(\neg x_1 + \neg w + y_3)$ can be derived.

4.2 Recursive Learning

Recursive learning has been extensively used in EDA [19]. Moreover, in [26] the recursive learning paradigm has been extended to CNF formulas. Next, we briefly describe how this can be done in practice.

For any clause ω in a CNF formula φ to be satisfied, at least one of its yet unassigned literals *must* be assigned value 1. Recursive learning on CNF formulas consists of studying the different ways of satisfying a given selected clause and identifying common assignments, which are then deemed *necessary* for the clause to become satisfied and consequently for the instance of SAT to be satisfiable. Clearly, and because conflict diagnosis can also be implemented, each identified assignment needs to be adequately *explained*. Consequently, with each identified assignment a clause that describes *why* the assignment is necessary is created. Let us consider the example CNF formula of Figure 4. In order to satisfy clause ω_3, either $w = 1$ or $y = 1$. Considering each assignment separately leads to the implied assignment $x = 1$; for $w = 1$ due to ω_1 and for $y = 1$ due to ω_2. Hence, the assignment $x = 1$ is necessary if the CNF formula is to be satisfied. One sufficient explanation for this implied assignment is given by the logical implication $(z = 1) \wedge (u = 0) \Rightarrow (x = 1)$, which can be represented in clausal form as $(\neg z + u + x)$. Consequently, this clause represents a new *implicate* of the Boolean function associated with the CNF formula and so it can be added to the CNF formula. This new clause also implies the assignment $x = 1$ as long as $z = 1$ and $u = 0$, as intended. As with recursive learning for combinational circuits, recursive learning for CNF formulas can be generalized to any recursion depth. The proposed recursive learning algorithm is further detailed in [26].

Observe that our proposed recursive learning procedure derives and *records* implicates of the function associated with the CNF formula. Clearly, these implicates prevent repeated derivation of the same assignments during the subsequent search. In contrast, the recursive learning procedure developed for combinational circuits [19] only records necessary assignments. Hence, when used as part of a search algorithm, the original recursive learning

procedure might eventually re-derive some of the already derived necessary assignments.

5 Solving SAT on Combinational Circuits

It is generally accepted that the utilization of CNF models and SAT algorithms has important advantages:

1. Existing, and extensively validated SAT algorithms, can be used instead of dedicated algorithms.
2. New improvements and new SAT algorithms can be easily applied to each target application.

In contrast, the utilization of CNF formulas and associated SAT algorithms is also characterized by several drawbacks:

1. As observed in [39], the structural information of the circuit, often of crucial importance, is lost.
2. In many EDA problems, a large number of instances of SAT has to be solved for each circuit. Hence, mapping a given problem description into SAT can represent a significant percentage of the overall running time [25].
3. Computed input patterns are in general overspecified. Overspecification can be a serious drawback in different applications, including circuit testing and binate constraint solving.

With the purpose of addressing these problems, in [39, 40] a new dynamic data structure, i.e. an extended implication graph, is proposed for solving instances of SAT in combinational circuits. Despite the promising results of [39, 40], utilizing a new data structure requires dedicated algorithms. Hence new search pruning techniques, developed for example in the context of SAT algorithms, will have to be adapted to the circuit graph data structure.

In this section we illustrate how to utilize structural information in SAT algorithms [37]. To a generic SAT algorithm we add a layer that maintains circuit-related information, e.g. fanin/fanout information as well as value justification relations. The proposed approach allows using any SAT algorithm to which this layer can be added. The main advantages of the proposed approach is that some of the previously mentioned drawbacks, i.e. inaccessibility to structural information and overspecification of input patterns, are eliminated. The main contribution over the work of [39] is that data structures used for SAT need not be modified, and so existing algorithmic solutions for SAT can naturally be augmented with the proposed layer for handling structural information. Moreover, the approach proposed in this paper is significantly simpler than the one in [39], since only minor modifications to SAT algorithms are required.

Let C_p denote a property of a combinational circuit C which is to be satisfied to an objective value o. This satisfiability problem is denoted by $\langle C_p, o \rangle$ and can be mapped into an instance of SAT, φ. The following information is associated with each variable x of φ, that also represents a circuit node x of C:

Gate	$\upsilon_0(x)$	$\upsilon_1(x)$
x = AND(w_1, ..., w_k)	1	$\|FI(x)\|$
x = NAND(w_1, ..., w_k)	$\|FI(x)\|$	1
x = NOR(w_1, ..., w_k)	1	$\|FI(x)\|$
x = XOR(w_1, ..., w_k)	$\|FI(x)\|$	$\|FI(x)\|$

Table 2: Threshold values on assigned inputs

Gate	$w_i = 0$	$w_i = 1$
x = AND(w_1, ..., w_k)	$\iota_0(x)$	$\iota_1(x)$
x = NAND(w_1, ..., w_k)	$\iota_1(x)$	$\iota_0(x)$
x = NOR(w_1, ..., w_k)	$\iota_1(x)$	$\iota_0(x)$
x = XOR(w_1, ..., w_k)	$\iota_0(x)$, $\iota_1(x)$	$\iota_0(x)$, $\iota_1(x)$

Table 3: Justification counters associated with gate inputs

1. $FI(x)$ denotes the fanin nodes of x.
2. $FO(x)$ denotes the set of fanout nodes of x.
3. $\upsilon_v(x)$ denotes the threshold value on the number of suitable assigned inputs (of x) that are necessary for justifying value v on node x.
4. $\iota_v(x)$ denotes the actual counter of assigned inputs (of x) that are involved in justifying the value v on node x.

Note that the value assigned to each variable x is denoted by $v(x)$. Moreover, observe that each circuit node x, with assigned value v, becomes justified whenever $\iota_v(x) \geq \upsilon_v(x)$.

Table 2 contains a few examples of threshold values on the number of assigned inputs required for justifying a given node. For example, for an AND gate at least one input assigned value 0 justifies the assignment of value 0 to x, whereas for value 1 all inputs must be assigned value 1. Hence, $\upsilon_0(x) = 1$ and $\upsilon_1(x) = |FI(x)|$. As another example, observe that for an XOR gate justification of any assigned value requires assignments to all gate inputs; hence $\upsilon_0(x) = \upsilon_1(x) = |FI(x)|$. For other simple gates this information can also be easily derived, and in all cases we have $\upsilon_0(x), \upsilon_1(x) \in \{1, |FI(x)|\}$.

For any simple gate with output x, we can associate with each fanin node w the counters that must be updated as the result of assigning a value v to w. For example, for an AND gate an assignment of 0 to a fanin node w increments $\iota_0(x)$ by 1, and an assignment of 1 to fanin node w increments $\iota_1(x)$ by 1. These relations are illustrated in Table 3 for a few example gates. Note that for the XOR gates, both counters are updated when an input node becomes assigned.

As with standard search algorithms in combinational circuits [1], a *justification frontier* is maintained, which denotes the sets of variables/nodes that require justification. Observe that the condition that indicates the need for node justification is $(v(x) = v) \wedge (\iota_v(x) < \upsilon_v(x))$, where

$v \in \{0, 1\}$.

Given the previous definitions, a SAT algorithm can be adapted so that the information regarding justification can be properly maintained. Moreover, the fanin information can be used for implementing structure-based heuristic decision making procedures, e.g. *simple or multiple backtracing* [1]. With respect to the algorithm of Figure 2, functions `Deduce()` and `Diagnose()` have to invoke dedicated procedures for updating node justification information. Additionally, the `Decide()` function now tests for satisfiability by checking for an empty justification frontier instead of checking whether all clauses are satisfied. These are the only required modifications to the general SAT algorithm. In addition, the `Decide()` function can optionally be modified to perform backtracing given the fanin information associated with each variable.

We should note that the data structures described above operate in much the same way as justification works in combinational circuits [1]. The main difference is that in our approach justification and value consistency are formally dissociated; value consistency is handled by the SAT algorithm and justification by the new added layer.

Moreover, we should observe that by taking into account the circuit structure information, the recursive learning procedure described in Section 4.2 can be made simpler, since only clause justifications of nodes in the fanin of a given unjustified node need to be considered.

6 Recent Work

Recent research work on SAT algorithms has included equivalency reasoning [21] and randomization with restarts [14]. Both approaches show great promise for EDA applications.

Equivalency reasoning targets the simplification of CNF formulas, either before or during the search, its main objective being the identification of equivalency clauses $(x + \neg y) \cdot (\neg x + y)$, that indicate that x and y must always be assigned the same value. Hence, variable y can be replaced by variable x, and one variable is eliminated.

Randomization [21] can be viewed as the process of introducing a certain degree of uncertainty in selecting branching variables and values during the search. The addition of randomization allows for repeatedly restarting the search each time a given limit number of decisions is reached. Restarts with randomization allow searching different regions of the search space and have been shown to yield dramatic improvements on satisfiable instances.

With respect to SAT algorithms for EDA applications, recent work has also attempted to exploit the fact that in many applications SAT solvers tend to be used iteratively and/or incrementally. Specific techniques for the iterative use of SAT algorithms [25] or the incremental formulation of problem instances [18] have been proposed.

Finally, the interest of the EDA community in solving SAT has led to the proposal of dedicated reconfigurable hardware architectures [2, 43] that, despite being significantly less sophisticated than software algorithms, can achieve significant speedups for specific classes of instances.

7 Conclusions

This paper surveys applications of SAT models to EDA applications, and briefly describes the core techniques that characterize modern SAT solvers, capable of solving large and hard instances of SAT. In addition, the paper describes different recently proposed techniques, that show promise for EDA applications. Among these, we address the extension of recursive learning to CNF formulas, adapting SAT solvers to combinational circuits, equivalency reasoning and randomization.

Despite the large number of practical EDA applications that can be mapped into SAT instances, and despite the improvements in the effectiveness of SAT algorithms, SAT is an NP-complete problem. Most (if not all) SAT solvers are still unable to solve many practical problem instances, many of these from EDA applications. The recent increase in the number of EDA applications utilizing SAT models further motivates a continuing effort towards improving SAT algorithms.

References

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design,* Computer Science Press, 1990.

[2] M. Abramovici, J. De Sousa, D. Saab, "A Massively-Parallel Easily-Scalable Satisfiability Solver Using Reconfigurable Hardware," to appear in *Proceedings of the Design Automation Conference*, June 1999.

[3] 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.

[4] R. Bayardo Jr. and R. Schrag, "Using CSP Look-Back Techniques to Solve Real-World SAT Instances," in *Proceedings of the National Conference on Artificial Intelligence (AAAI-97)*, 1997.

[5] A. Biere, A. Cimatti, E. Clarke and Y. Zhu, "Symbolic Model Checking without BDDs," in *Proceedings of TACAS*, LNCS 1579, March 1999.

[6] M.N. Velev and R.E. Bryant, "Superscalar Processor Verification Using Reductions of the Logic of Equality with Uninterpreted Functions to Propositional Logic," in *Proceedings of CHARME,* LNCS 1703, September 1999.

[7] C.-A. Chen and S. K. Gupta, "A Satisfiability-Based Test Generator for Path Delay Faults in Combinational Circuits," in *Proceedings of the Design Automation Conference*, June 1996.

[8] P. Chen and K. Keutzer, "Towards True Crosstalk Noise Analysis," to appear in *Proceedings of the International Conference on Computer-Aided Design*, November 1999.

[9] O. Coudert, "On Solving Covering Problems," in *Proceedings of the Design Automation Conference*, June 1996.

[10] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *Journal of the Association for Computing Machinery*, vol. 7, pp. 201-215, 1960.

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

[12] L. Entrena and K.-T. Cheng, "Sequential Logic Optimization by Redundancy Addition and Removal," in *Proceedings of the International Conference on Computer-Aided Design*, November 1993.

[13] F. Fallah, S. Devadas and K. Keutzer, "Functional Vector Generation For HDL Models Using Linear Programming and 3-Satisfiability," in *Proceedings of the Design Automation Conference*, June 1998.

[14] C. P. Gomes, B. Selman and H. Kautz, "Boosting Combinatorial Search Through Randomization," in *Proceedings of the National Conference on Artificial Intelligence*, July 1998.

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

[16] A. Gupta and P. Ashar, "Integrating a Boolean Satisfiability Checker and BDDs for Combinational Equivalence Checking," in *Proceedings of the International Conference in VLSI Design*, January 1998.

[17] J. Kim, J. Marques-Silva, H. Savoj and K. A. Sakallah, "RID-GRASP: Redundancy Identification and Removal Using GRASP", in *International Workshop on Logic Synthesis*, May 1997.

[18] J. Kim, J. Whittemore, J. Marques-Silva and Karem A. Sakallah, "On Applying Incremental Satisfiability to Delay Fault Testing," to appear in *Proceedings of the Design, Automation and Test in Europe Conference*, March 2000.

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

[20] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 1, pp. 4-15, January 1992.

[21] C.-M. Li, "Equivalency Reasoning to Solve a Class of Hard SAT Problems," *Submitted for publication*. (Available from http://www.research.att.com/~kautz/challenge/cli.ps.)

[22] V. Manquinho, A. Oliveira and J. Marques-Silva, "Models and Algorithms for Computing Minimum-Size Prime Implicants," in *Proceedings of the International Workshop on Boolean Problems*, September 1998.

[23] V. Manquinho and J. Marques-Silva, "On Using Satisfiability-Based Pruning Techniques in Covering Algorithms," in *Proceedings of the Design, Automation and Test in Europe Conference*, March 2000.

[24] J. Marques-Silva and K. A. Sakallah, "GRASP—A New Search Algorithm for Satisfiability," in *Proceedings of the International Conference on Computer-Aided Design*, November 1996.

[25] J. Marques-Silva and K. A. Sakallah, "Robust Search Algorithms for Test Pattern Generation," in *Proceedings of the Fault-Tolerant Computing Symposium*, June 1997.

[26] J. Marques-Silva and T. Glass, "Combinational Equivalence Checking Using Satisfiability and Recursive Learning," in *Proceedings of the Design and Test in Europe Conference*, March 1999.

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

[28] P. McGeer, A. Saldanha, P. R. Stephan, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Timing Analysis and Delay-Test Generation Using Path Recursive Functions," in *Proceedings of the International Conference on Computer-Aided Design*, November 1991.

[29] G.-J. Nam, K. A. Sakallah and R. Rutenbar, "Satisfiability Based FPGA Routing," in *Proceedings of the International Conference on VLSI Design*, January 1999.

[30] G.-J. Nam, K. A. Sakallah, and R. A. Rutenbar, "Satisfiability-Based Layout Revisited: Detailed Routing of Complex FPGAs Via Search-Based Boolean SAT," in *International Symposium on Field-Programmable Gate Arrays*, February 1999.

[31] A. Saldanha and V. Singhal, "Solving Satisfiability in CAD Problems," in *Proceedings of the Cadence Technical Conference*, May 1997.

[32] B. Selman, H. Levesque and D. Mitchell, "A New Method for Solving Hard Satisfiability Problems," in *Proceedings of the National Conference on Artificial Intelligence (AAAI-92)*, pp. 440-446, 1992.

[33] Y. Shang and B. W. Wah, "A Discrete Lagrangian-Based Global-Search Method for Solving Satisfiability Problems," Journal of Global Optimization, Jan. 1998.

[34] M. Sheeran and G. Stalmarck, "A Tutorial on Stalmarck's Proof Procedure for Propositional Logic," in *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, November 1998.

[35] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, 1995.

[36] L. G. Silva, J. P. Marques-Silva, L. M. Silveira and K. A. Sakallah, "Satisfiability Models and Algorithms for Circuit Delay Computation," in *Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, December 1997.

[37] L. G. Silva, L. Silveira and J. Marques-Silva, Algorithms for Solving Boolean Satisfiability in Combinational Circuits, in *Proceedings of the Design and Test in Europe Conference*, March 1999.

[38] P. R. Stephan, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Combinational Test Generation Using Satisfiability," *IEEE Transactions on Computer-Aided Design*, vol. 15, no. 9, pp. 1167-1176, September 1996.

[39] P. Tafertshofer, A. Ganz and M. Henftling, "A SAT-Based Implication Engine for Efficient ATPG, Equivalence Checking, and Optimization of Netlists," in *Proc. of the Int'l Conf. on Computer-Aided Design*, pp. 648-657, November 1997.

[40] P. Tafertshofer e A. Ganz, "SAT Based ATPG Using Fast Justification and Propagation in the Implication Graph," in *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design* (ICCAD), November 1999.

[41] R. Zabih and D. A. McAllester, "A Rearrangement Search Strategy for Determining Propositional Satisfiability," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 155-160, 1988.

[42] H. Zhang, "SATO: An Efficient Propositional Prover," in *Proceedings of International Conference on Automated Deduction*, July 1997.

[43] P. Zhong, P. Ashar, S. Malik and M. Martonosi, "Using Reconfigurable Computing Techniques to Accelerate Problems in the CAD Domain: A Case Study with Boolean Satisfiability," in *Proceedings of the Design Automation Conference*, June 1998.