Algorithms for Solving Boolean Satisfiability in Combinational Circuits

Luís Guerra e Silva, L. Miguel Silveira and João Marques-Silva

Instituto Superior Técnico
Cadence European Labs/INESC
1000 Lisboa, Portugal
e-mail: {lgs,lms,jpms}@algos.inesc.pt

Abstract

Boolean Satisfiability is a ubiquitous modeling tool in Electronic Design Automation, It finds application in test pattern generation, delay-fault testing, combinational equivalence checking and circuit delay computation, among many other problems. Moreover, Boolean Satisfiability is in the core of algorithms for solving Binate Covering Problems. This paper describes how Boolean Satisfiability algorithms can take circuit structure into account when solving instances derived from combinational circuits. Potential advantages include smaller run times, the utilization of circuit-specific search pruning techniques, avoiding the overspecification problem that characterizes Boolean Satisfiability testers, and reducing the time for iteratively generating instances of SAT from circuits. The experimental results obtained on several benchmark examples in two different problem domains display dramatic reductions in the run times of the algorithms, and provide clear evidence that computed solutions can have significantly less specified variable assignments than those obtained with common SAT algorithms.

1. Introduction

Boolean Satisfiability (SAT) is intrinsic to many problems in Electronic Design Automation (EDA). Originally motivated by the work of T. Larrabee in test pattern generation [15], SAT models and techniques have since been applied to delay-fault testing, equivalence checking, circuit delay computation, logic synthesis and functional vector generation [8], among other applications. (See [5, 7, 15, 17, 18, 20] for an overview of applications of SAT to EDA.) Moreover, SAT can also play a central role in solving instances of binate covering problems (BCP) [6, 9, 10, 11, 13], in particular for those in which the constraints are hard to satisfy, e.g. in computing minimum size test patterns [10]. SAT also plays a key role in other domains, including for example Artificial Intelligence [3, 21] and Operations Research [2]. Recent years have seen dramatic improvements in SAT algorithms, which have been thoroughly validated in different application areas [3, 16, 21].

With respect to applications of SAT in EDA, in most cases the original problem formulation starts from a circuit description, for which a given (circuit) property needs to be validated for at least one primary input vector. The resulting circuit formulation, which may only be implicitly specified, is then mapped into an instance of SAT, in most cases using Conjunctive Normal Form (CNF) formulas.

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.

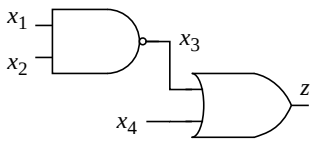
In contrast, the utilization of CNF formulas and associated SAT algorithms is also characterized by several drawbacks:

1. As observed in [20], 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 [15].
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 [20] 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 [20], 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 paper we show how to utilize structural information in SAT algorithms. 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 [20] 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 [20], since only minor modifications to SAT algorithms are required.

The paper is organized as follows. Section 2 introduces basic definitions associated with SAT and combinational circuits. Next we briefly survey SAT algorithms, giving particular emphasis to those that have been shown to be effective in solving EDA problems. Afterwards, in Section



$$\varphi = (x_1 + x_3) \cdot (x_2 + x_3) \cdot \left(\bar{x}_1 + \bar{x}_2 + \bar{x}_3\right) \cdot \left(\bar{x}_3 + z\right) \cdot \left(\bar{x}_4 + z\right) \cdot \left(x_3 + x_4 + \bar{z}\right)$$

(a) Consistent assignments

$$\varphi' = (x_1 + x_3) \cdot (x_2 + x_3) \cdot \left(\bar{x}_1 + \bar{x}_2 + \bar{x}_3\right) \cdot \left(\bar{x}_3 + z\right) \cdot \left(\bar{x}_4 + z\right) \cdot \left(x_3 + x_4 + \bar{z}\right) \cdot (\bar{z})$$

(b) With property $z = 0$

Figure 1: Example circuit and CNF formula

4, we detail the proposed approach for taking into consideration structural information while solving SAT. Section 5 analyzes preliminary results on two EDA applications. The paper concludes in Section 6 by reviewing the contributions and providing some perspective on future research work.

2. Definitions

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 can be found for example in [15, 17].) If we 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. Examples include the CNF formulas for test pattern generation [15] and circuit delay computation expressions [18].

3. Boolean Satisfiability Algorithms

The overall organization of a generic SAT algorithm is shown in Figure 2. This generic SAT algorithm captures the organization of several of the most competitive algorithms [3, 16, 21].

The algorithm conducts 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 [3, 16]. 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 [3, 16, 21] are characterized by several of the following key properties:

```
// 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

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 [3, 16, 21].
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 [3, 16].
3. Other techniques have been developed. *Relevance-Based Learning* [3] extends the life-span of large recorded clauses that will eventually be deleted. *Conflict-Induced Necessary Assignments* [16] 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 [16]. This in general is denoted by a `Preprocess()` function.

4. Satisfiability in Combinational Circuits

4.1. Additional Data Structures

Let C_p denote a property of a combinational circuit C which is to be satisfied to an objective value o and which can be described by a given set of clauses. 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 1: 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)	$\iota_0(x)$ and $\iota_1(x)$	$\iota_0(x)$ and $\iota_1(x)$

Table 2: 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 $\nu(x)$. Moreover, observe that each circuit node x, with assigned value v, becomes justified whenever $\iota_v(x) \geq \upsilon_v(x)$.

Table 1 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 2 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 $(\nu(x) = v) \wedge (\iota_v(x) < \upsilon_v(x))$, where $v \in \{0, 1\}$.

4.2. Modifications to the SAT Algorithm

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, and hence conflicts, are handled by the SAT algorithm, and justification by the new added layer.

4.3. Handling Special Implications

Besides taking structural information into account, further improvements are possible when the SAT algorithm is intended to prove or disprove a given circuit property, and when several constraints are formulated as logical implications. This technique can be viewed as a generalization of *syntactic satisfiability* [17]. It is plain that for a combinational circuit we can find consistent assignments. Syntactic satisfiability hinges on this fact to allow the search algorithm to stop the search when clauses not yet satisfied are guaranteed to have a satisfying assignment, e.g. when no clause of the original circuit CNF formula has literals assigned value 0.

Let us suppose that when proving a given circuit property we have several implications of the form $y \rightarrow \sum w_i$, where variable y is not associated with one of the circuit variables, being used instead for describing the target circuit property. This implication is satisfied if $y = 0$. Hence, only when $y = 1$ is a necessary assignment, do we need to require that the equivalent clausal form $\left(\bar{y} + \sum w_i\right)$ be satisfied. We refer to this conditional consideration of clauses as *implicational syntactic satisfiability.*

4.4. Additional Advantages

For some EDA problems it is necessary to repeatedly solve instances of SAT. *Pervasive clauses* were introduced in [17] for denoting a clause that is recorded while solving an instance of SAT, and which can be used subsequently for solving other instances of SAT that are associated with the same circuit. For example, pervasive clauses may be used in test pattern generation (TPG) for denoting value relations between circuit nodes.

With the structural information described in this section, we can also characterize which variables can yield pervasive clauses and which cannot. Hence, while solving a given EDA problem, the SAT algorithm can identify pervasive clauses, which it can subsequently re-utilize while

Circuit	#PI	#PO	#G	TPG			CDC	
				#F	#D	#R	LTP	Δ
C432	36	7	160	524	520	4	17	17
C499	41	32	202	758	750	8	11	11
C880	60	26	383	942	942	0	24	24
C1355	41	32	546	1574	1566	8	24	24
C1908	33	25	880	1879	1870	9	40	37
C2670	233	140	1193	2747	2630	117	32	30
C3540	50	22	1669	3428	3291	137	47	46
C5315	178	123	2307	5350	5291	59	49	47
C6288	32	32	2406	7744	7710	34	124	123
C7552	207	108	3512	7550	7419	131	43	42

Table 3: Statistics for ISCAS'85 circuits

Circuit	CNF(s)	GRASP				CGRASP			
		#B	#NCB	%A	SAT(s)	#B	#NCB	%A	SAT(s)
C432	3.24	2855	115	100	7.54	167	8	75	2.17
C499	5.11	1254	1110	100	13.07	24	24	82	3.02
C880	4.89	2690	968	99	30.21	55	31	33	3.12
C1355	27.92	5464	644	100	77.383	73	40	89	18.87
C1908	27.56	3505	2224	100	108.00	1353	946	80	33.57
C2670	13.97	12949	6997	85	437.17	5132	1296	26	34.20
C3540	66.40	33049	12568	99	839.00	692	199	66	83.60
C5315	32.79	5081	3138	98	2727.68	1671	631	22	80.24
C6288	362.00	149374	5360	100	4215.33	16314	398	77	467.23
C7552	65.18	70958	35322	98	12641.35	4811	1704	44	247.95

Table 4: Results for test pattern generation

solving instances that are associated with the same circuit.

5. Experimental Results

In this section we evaluate the practical usefulness of the circuit structure-aware SAT algorithm described in Section 4. For this purpose, we used a state of the art public-domain SAT algorithm, GRASP [16], and built on top of this algorithm a new SAT algorithm that takes structural information into account, CGRASP. Two EDA applications are considered for comparing GRASP and CGRASP, namely test pattern generation [15, 17] and circuit delay computation (CDC) [7, 18]. Statistics for the ISCAS'85 benchmark circuits [4], regarding TPG and circuit delay computation, are shown in Table 3. #PI, #PO, #G, #F, #D, #R, LPT and Δ denote, respectively, the number of primary inputs, the number of primary outputs, the number of gates, the number of stuck-at faults, the number of detectable faults, the number of redundant faults, the longest topological delay and the critical circuit delay under floating-mode operation [7]. All the experimental results shown below were obtained on a P-II 266 MHz Linux workstation with 128 MByte of physical memory.

5.1. Test Pattern Generation

The first experiment consists in evaluating CGRASP for TPG, using a simplified version of the model in [17]. For this purpose, both GRASP and CGRASP were run on the ISCAS'85 benchmark circuits. The results are shown in Table 4. For each algorithm, CNF(s), #B, #NCB, %A and SAT(s) denote, respectively the total CNF formula build time, the total number of backtracks, the total number of non-chronological backtracks, the average percentage of assigned variables over all solved instances and the SAT search time. In this experiment, each fault was individually targeted, i.e. fault simulation and hence fault dropping were not applied. Moreover, neither GRASP nor CGRASP abort any fault. As can be readily concluded, the search times become drastically reduced when structural information is taken into account, and a justification frontier is used for SAT purposes. Indeed, for some of the benchmark circuits the search times can be reduced by almost two orders of magnitude.

Circuit	#PI	#PO	#G	LTP	Δ
csa.32.16	65	33	170	69	66
csa.32.8	65	33	180	73	38
csa.32.4	65	33	200	81	30
csa.64.16	129	65	340	137	70
csa.64.8	129	65	360	145	46
csa.64.4	129	65	400	161	46
csa.128.16	257	129	680	273	78
csa.128.8	257	129	720	289	62
csa.128.4	257	129	800	321	78

Table 5: Statistics for carry-skip adders

Besides the reduction in CPU times from GRASP to CGRASP, we can also observe similar reductions in the total number of backtracks. Nevertheless, the pruning techniques of GRASP are still used in CGRASP, as the number of non-chronological backtracks clearly illustrates. Moreover, CGRASP computes test patterns that are significantly less specified than the test patterns computed by GRASP (see Table 4). Hence, by taking structural information into account we can effectively handle the overspecification problem, and compute test patterns that can be as specified as those obtained with purely structural methods [1]. Given the experimental analysis of [19], where SAT-based ATPG algorithms are shown to be competitive with structural ATPG methods, and the above experimental results, we hypothesize that further improvements to SAT-based ATPG algorithms are possible.

5.2. Circuit Delay Computation

Another potential application is SAT-based circuit delay computation [18, 12]. Experimental results comparing GRASP and CGRASP for circuit delay computation are shown in Table 6. For these experiments, the model of [18, 12] is used, and unit gate delays are assumed. In addition to the ISCAS'85 circuits, we generated several carry-skip adders[1], to evaluate how each algorithm performs with increasing circuit size and complexity. As can be concluded once more, utilizing to the structural information of the circuit proves crucial in reducing the CPU times spent searching. Similarly to TPG, the reduction in run times can be dramatic, in some cases two orders of magnitude reductions are observed. Once more, despite the very significant reductions in the run times, we can still observe the pruning techniques of GRASP being used. As shown, the number of non-chronological backtracks still represents a significant percentage of the overall number of

1. See Table 5 for statistics regarding these circuits.

Circuit	CNF(s)	GRASP			CGRASP		
		#B	#NCB	SAT(s)	#B	#NCB	SAT(s)
C432	0.01	66	3	0.09	0	0	0.01
C499	0.01	0	0	0.51	0	0	0.01
C880	0.01	20	12	0.30	0	0	0.02
C1355	0.05	1540	397	9.03	1	1	0.13
C1908	0.12	93	67	9.30	56	42	1.31
C2670	0.11	558	231	21.88	422	177	2.17
C3540	0.06	41	30	3.92	3	3	0.49
C5315	0.06	35	28	8.99	17	2	0.23
C6288	0.14	1216	223	37.34	1725	208	33.96
C7552	0.06	8	8	23.79	1	1	0.47
csa.32.16	0.02	0	0	0.17	0	0	0.01
csa.32.8	0.23	44	19	1.39	0	0	0.22
csa.32.4	1.32	431	240	7.38	210	33	0.83
csa.64.16	1.21	104	47	10.72	0	0	1.04
csa.64.8	6.01	1745	425	80.01	694	57	4.66
csa.64.4	17.81	7994	3058	414.84	3510	1713	19.52
csa.128.16	36.24	15110	2500	1446.63	2526	105	31.42
csa.128.8	89.48	56570	15122	5853.47	11053	3665	93.17
csa.128.4	199.15	108098	44309	24725.92	33190	23238	399.57

Table 6: Results for circuit delay computation

backtracks. Moreover, as was noted for TPG, the advantages of CGRASP become patent with increasing circuit sizes, especially for the carry-skip adders [18, 12]. We should observe that one of the conclusions of [12] is that GRASP is one of the most competitive SAT algorithms for computing circuit delays. Hence, the above results clearly indicate that by taking the structural information into account, SAT solvers can be significantly improved upon.

6. Conclusions

This paper proposes a new algorithm for solving Boolean Satisfiability problems in combinational circuits. For manipulating structural information, the proposed approach requires minor modifications in existing SAT algorithms. The experimental evaluation of the new algorithm on different applications shows dramatic improvements over other Boolean Satisfiability solvers that have been shown to be highly effective [16]. As shown in the previous sections, the proposed algorithm also effectively addresses the overspecification problem, leading to significant reductions in the number of assigned variables for the satisfiable instances of SAT.

Besides the computational improvements observed, the proposed approach only requires minor modifications to existing SAT algorithms, allowing new algorithmic improvements to be readily applied to solving SAT in combinational circuits. This approach is in clear contrast to the one in [20], which requires dedicated data structures and associated algorithms.

One potential drawback of the proposed algorithm, that is also common to all SAT-based approaches, is that the CNF formula of the circuit property to prove needs to be generated each time, thus impacting overall efficiency. In the examples we ran, the time to generate the CNF formulas can range from a small percentage, for the larger benchmarks in TPG, to a large percentage, for the larger examples in CDC and the smaller examples in TPG. Future research work will necessary address this issue.

Other future improvements involve incorporating additional structural pruning techniques, specific to combinational circuits, with the objective of further simplifying solving SAT in circuits. Examples include application-specific pruning techniques, e.g. unique sensitization points [15], that can reduce the amount of search for each application.

References

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

[2] P. Barth, "A Davis-Putnam Enumeration Algorithm for Linear pseudo-Boolean Optimization," Technical Report MPI-I-95-2-003, Max Planck Institute for Computer Science, 1995.

[3] R. Bayardo Jr. and R. Schrag, "Using CSP Look-Back Techniques to Solve Real-World SAT Instances," in *Proc. of the Nat'l Conf. on Artificial Intelligence*, pp. 203-208, July 1997.

[4] F. Brglez and H. Fujiwara, "A Neutral List of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN," in *Proc. of the Int'l Symp. on Circuits and Systems*, 1985.

[5] C.-A. Chen and S. K. Gupta, "A Satisfiability-Based Test Generator for Path Delay Faults in Combinational Circuits," in *Proc. of the Design Automation Conf.*, pp. 209-214, June 1996.

[6] O. Coudert, "On Solving Covering Problems," in *Proc. of the Design Automation Conf.*, June 1996.

[7] S. Devadas, K. Keutzer and S. Malik, "Computation of Floating Mode Delay in Combinational Circuits: Practice and Implementation," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 12 no. 12, pp. 1923-1936, December 1993.

[8] F. Fallah, S. Devadas and K. Keutzer, "Functional Vector Generation For HDL Models Using Linear Programming and 3-Satisfiability", in *Proc. of the Design Automation Conf.*, pp. 528-533, June 1998.

[9] F. Ferrandi et al., "Symbolic Algorithms for Layout-Oriented Synthesis of Pass Transistor Logic Circuits," in *Proc. of the Int'l Conf.e on Computer-Aided Design*, November 1998.

[10] P. Flores, H. Neto and J. Marques-Silva, "An Exact Solution to the Minimum-Size Test Pattern Problem" in *Proc. of the Int'l Conference on Computer Design*, October 1998.

[11] R. Fuhrer and S. Nowick, "Exact Optimal State Minimization for 2-Level Output Logic," in *Int'l Workshop on Logic Synthesis*, June 1998.

[12] L. Guerra e Silva, J. 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 (TAU)*, December 1997.

[13] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers, 1996.

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

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

[16] J. Marques-Silva and K. A. Sakallah, "GRASP—A New Search Algorithm for Satisfiability," in *Proc. of the Int'l Conf. on Computer-Aided Design*, pp. 220-227, November 1996. (URL: http://algos.inesc.pt/grasp/grasp.tar.gz.)

[17] J. Marques-Silva and K. A. Sakallah, "Robust Search Algorithms for Test Pattern Generation," in *Proc. of the Int'l Symp. on Fault-Tolerant Computing*, pp. 152-161, June 1997.

[18] 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 *Proc. of the Int'l Conf. on Computer-Aided Design*, pp. 180-183, November 1991.

[19] P. Stephan, R.K. Brayton and A.L. Sangiovanni-Vincentelli, "Combinatorial Test Generation Using Satisfiability", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, September 1996.

[20] 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.

[21] H. Zhang, "SATO: An Efficient Propositional Prover," in *Proc. of Int'l Conf. on Automated Deduction*, pp. 272-275, July 1997.