On Properties of SAT Instances Produced by SAT-Based ATPGs

Jiří Balcárek[1], Petr Fišer[2], and Jan Schmidt[2]

[1] Czech Technical University in Prague, Faculty of Electrical Engineering,
Karlovo nám. 13, CZ-121 35, Prague 2, Czech Rep.
balcaj3@fel.cvut.cz,
[2] Czech Technical University in Prague, Faculty of Information Technology,
Kolejní 550/2, CZ-160 00, Prague 6, Czech Rep.
fiserp@fit.cvut.cz, schmidt@fit.cvut.cz

Abstract. In this paper we propose a study of properties of SAT (satisfiability) instances produced by SAT-based Automatic Test Pattern Generators (ATPGs). Standard non-commercial SAT solvers are being widely used for the purpose of solving these instances. We show an analysis of properties of these special SAT instances. Even though these ATPG SAT instances have been thoroughly studied in the past, we show some newly found properties. Particularly, reasons why ATPG SAT instances are 'easy to be solved' are shown by analysis of the SAT instances. Then, unexpected behavior of ATPG SAT instances, in terms of their satisfiability, was observed. Next, we propose solution-preserving SAT transformations and study the properties of the reduced SAT instances.

1 Introduction

Testing has nowadays become an integral part of the chip manufacturing process. Test patterns are applied to the chip and its response is evaluated, in order to detect most of the chip's faults. Millions of test vectors need to be applied to the chip to test it sufficiently now. Thus, the manufacture test application costs some time and, consequently money. Therefore designers try to reduce the test cost by introducing design-for-testability features and Built-in Self-Test structures [1]. Anyway, the bandwidth of the datapath between the tester and the tested circuit still persists to be a bottleneck. Thus, the need for compression of the test patterns is inevitable. Sophisticated on-chip test decompression mechanisms [2], [3] have been developed for this purpose.

Most of the state-of-the art and commercial Automatic Test Pattern Generation (ATPG) tools are based on Satisfiability (SAT) solving [4], [5], [6]. These SAT-based ATPGs are popular due to their speed, enabling them to generate test patterns for extremely large circuits.

The NP-completeness of the SAT problem is well known [7]. Thus, there arises an apparent question why the SAT based ATPGs are as fast as reported [4], [5], even though tens of thousands SAT instances having tens of thousands variables are solved in the process. This issue has been studied in the past and it has been

concluded that ATPG SAT problems are 'easy' [8]. The expected time simplicity of solving these SATs has been proven for a particular (yet common) SAT solving procedure. The SAT solver running time as a function of the circuit cut-with was derived. Since it was shown that the circuit cut-width grows logarithmically with the circuit size, the authors expect no significant runtime growth with a growing circuit size. In contrast to [8], we study the very SAT instances, not the circuits properties and we show more convincing reasons for 'easiness' of the ATPG SAT instances. We investigate how the ATPG SAT instances differ from randomly generated SAT instances of the same parameters. Next, we study an interesting observation in the satisfiability of these instances.

We also focus on the number of the SAT solutions. In standard SAT-based ATPGs, only one SAT solution is needed. However, we proposed an application of SAT-based ATPGs for test pattern compression, which directly generates compressed test patterns in the process, for details see [13]. The algorithm requires considering more, or better all, solutions of each SAT instance. Standard Davis&Putnam (D&P)-based SAT solvers [9] cannot be efficiently used, as they produce only one SAT solution. More general ([14], [15]) SAT solvers must be used. This makes the ATPG SAT problem solving more difficult in contrast to problems of standard SAT-based ATPGs. We investigate possibilities of solutions-preserving SAT transformations, to maximally simplify the SAT instances.

1.1 Boolean Satisfiability

Let us have a Boolean formula given in a *Conjunctive Normal Form* (CNF). The CNF is a conjunction of disjunctions of literals. A literal is a variable or its complement. The disjunctions of literals are also called *clauses*.

Example 1. Let us have a formula in CNF: $\varphi = (a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a}) \wedge (a \vee c)$
The formula is called *satisfiable*, when there exists an assignment of variables a, b, c, so that the formula is equal to 1. We can see that φ is equal to one for a combination $(a, b, c) = (0, 0, 1)$, thus it is satisfiable.

Definition 1. *Deciding whether a formula in CNF is satisfiable is called a satisfiability (SAT) problem. If the formula consists only of clauses having exactly K literals, we call the problem K-SAT.*

It is well known that the K-SAT problem is NP-complete for $K \geq 3$ [7]. The 2-SAT problem is solvable in a polynomial time [17].

Definition 2. *Deciding on satisfiability of CNF formulae having a mixture of 2- and 3-literal clauses is called a 2+p-SAT problem [12]. Here p stands for a percentage of 3-literal clauses. This model interpolates between 2-SAT (for $p = 0$) and 3-SAT (for $p = 1$) problems. For $p > 0$ the 2+p-SAT problem is NP-complete as well.*

1.2 SAT-Based ATPGs

The *Automatic Test Pattern Generation* (ATPG) process consists in finding a set of *test patterns*, that are to be applied to the tested circuit's inputs, in order to detect possibly all faults (under a given fault model). Combinational circuits and a stuck-at fault model [1] will be assumed in the following text.

There were developed many ATPG algorithms in the past twenty years [18], [19], [20]. Lately, due to emergence of very fast SAT solvers, SAT-based ATPGs are becoming more and more popular [4], [5]. In principle, SAT-based ATPGs transform the fault-free circuit together with a given fault into a SAT instance. A solution of this instance describes a test pattern detecting the respective fault.

Note 1. In practice, only gates on paths exciting and propagating the tested fault are considered for the CNF construction. In other words, primary inputs, internal signals, and gates whose values do not affect detectability of the fault are omitted in the CNF.

Example 2. The main idea of the circuit-to-SAT conversion is shown in Fig. 1. A circuit of three gates is shown there. To detect a given fault, two copies of the circuit are created: the fault-free circuit and the circuit with a fault. In order to detect the fault, output values of the two circuits must differ. This is indicated by XORing the two outputs (X and X'). Each gate is described by its characteristic function in CNF and the CNF of the whole circuit is constructed as a conjunction of these characteristic functions. As a result, the final CNF will be satisfied for all assignments of values of primary input variables detecting the tested fault (together with values of internal signals). Characteristic functions

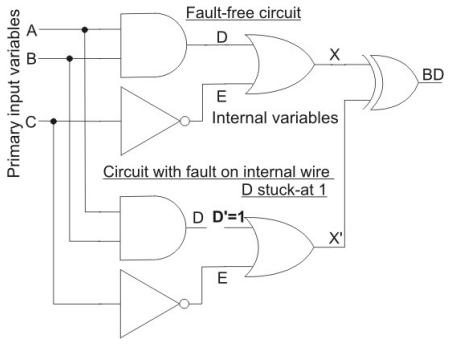

Fig. 1. Faulty and fault-free circuit and its CNF example [5]

are derived from logic functions of the gates. For example, let us consider the AND gate $D = A \wedge B$. For any two functions P and Q, $P = Q$ is equivalent to $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$. In this way the AND gate characteristic function is constructed as $(D \Rightarrow (A \wedge B)) \wedge ((A \wedge B) \Rightarrow D)$. Next we transform this expression to CNF, obtaining $(\overline{D} \vee A) \wedge (\overline{D} \vee B) \wedge (D \vee \overline{A} \vee \overline{B})$. All gates in Fig. 1

are processed in this way, to obtain the final CNF for detecting the stuck-at-1 fault on D:

$$(\overline{D} \vee A) \wedge (\overline{D} \vee B) \wedge (D \vee \overline{A} \vee \overline{B}) \wedge (C \vee E) \wedge (\overline{C} \vee \overline{E}) \wedge (X \vee \overline{D})$$
$$\wedge (X \vee \overline{E}) \wedge (\overline{X} \vee D \vee E) \wedge (D') \wedge (X' \vee \overline{D'}) \wedge (X' \vee \overline{E}) \wedge (\overline{X'} \vee D' \vee E)$$
$$\wedge (C \vee E) \wedge (\overline{C} \vee \overline{E}) \wedge (\overline{X} \vee X' \vee BD) \wedge (X \vee \overline{X'} \vee BD) \wedge (X \vee X' \vee \overline{BD})$$
$$\wedge (\overline{X} \vee \overline{X'} \vee \overline{BD}) \quad (1)$$

If there exists an assignment of variables for which this CNF is satisfied, the fault is detectable. The assignment of input variables corresponds to the test pattern detecting this fault. In particular, the example CNF (1) is satisfied by assignment $A = 1, B = 0, C = 1, D = 0, D' = 1, E = 0, X = 0, X' = 1, BD = 1$. Therefore, the stuck-at-1 at D fault is detected by the pattern $(A, B, C) = (101)$.

2 Random SATs vs. ATPG SATs

The difficulty of solving random SAT problems has been thoroughly studied in the past years. In [10] Selman proposed a metrics of 'difficulty' of solving the SAT problem, in terms of the number of the Davis&Putnam [11] algorithm steps. This algorithm has been established as a basis of most modern SAT solvers [9], [21]. He has found that there exists a phase transition, where SAT instances rapidly turn from satisfiable to unsatisfiable. At this transition (threshold) D&P-based algorithms suffer from extremely long solving time. Such a threshold was observed at the clauses/variables ratio near 4.3 for 3-SAT, independently of the number of variables. At this threshold, approximately 50% of random instances are satisfiable. An interesting phenomenon is observed for a 2-SAT problem: the SAT/UNSAT transition is continuous, the 50% satisfiability threshold lies near the ratio of 1.0, whereas there is no apparent solving time increase near this threshold. It was shown that 2+p-SAT problems behave like 2-SATs for $p < 0.4$ and like 3-SAT for p higher [12].

By processing approx. 60 circuits from the ISCAS [22], [23] and ITC'99 [24] benchmark sets, we have generated more than 180,000 ATPG SAT instances. By analyzing them we obtained some information on their properties. First, we analyzed the ratios of clauses of a given number of literals. From the nature of the circuit-to-SAT conversion it is expected that 2-literal and 3-literal clauses will prevail. This fact was more or less confirmed. The measured average percentages of K-literal clauses were as follows: 3% of 1-literal clauses, 70% of 2-literal ones, 24% of 3-literal clauses, and 2% of 4-literal ones. Less than 1% of 5 and more literal clauses was present. The distribution of 2-literal and 3-literal clauses is illustrated in Fig. 2. 1000 random SAT instances were tested and numbers of satisfiable instances were counted. Judging from this data we can say that the ATPG SAT instances are similar to 2+p-SATs, where $p = 0.24$ [12]. In order to judge on 'hardness' of these ATPG SATs we have computed the clauses/terms ratio, in order to cope with the above-mentioned metrics. The results were surprising: the average ratio was 2.37, ranging from 1.38 to 3.01.

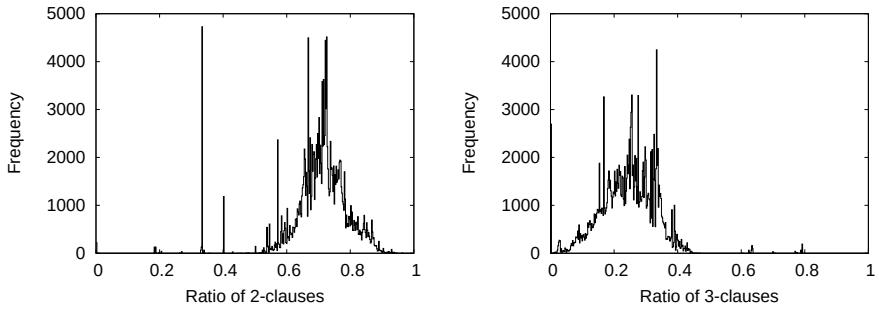

Fig. 2. Distribution of 2- and 3-literal clauses in ATPG SATs

Recall that the satisfiability threshold for 2-SAT or a similar 2+p-SAT is near the ratio of 1.0. I.e., random instances having such clauses/terms ratio should be mostly unsatisfiable, whereas the ATPG SATs are mostly satisfiable (99% in our measurements), since most of faults in the tested circuits are detectable. To justify this surprising fact, we have generated random instances with characteristics exactly like described in this paragraph, with constantly 300 variables, while we varied the number of clauses. Experimental results have confirmed the theory: all instances having the ratio above 1.8 were unsatisfiable. Here numbers of satisfiable formulas of 1000 random ones were measured. We have also generated random 'clones' of the real instances, by exchanging real variables in SAT by random ones. Only 10% of them were satisfiable. This brings us to the first conclusion: *ATPG SAT instances do differ from random instances; they are satisfiable, even though they shouldn't be.* Up to now, we have no fully relevant clue to explain this phenomenon. We have measured the *connectivity* of the SAT instances, when expressed as graphs [27]. However, we haven't found a significant difference between ATPG and random SAT instances of equal parameters.

3 Test Pattern Compression

Recently we have investigated test pattern compression methods based on finding the best test pattern overlap [2], [25]. Even though these methods use sophisticated compression algorithms, the compression ratio is limited by the structure of the pre-generated test patterns. Recently we have proposed a method based on an implicit representation of test patterns[13]. Such an implicit test representation, like CNF, allows for expressing all test patterns for each fault, without excessive memory demands. Since we are not limited by pre-generated test patterns, we can theoretically reach the best test compression ratio possible. For this purpose, a SAT solver producing more than one solution is required. This problem is referred to as an *#SAT* or *All-SAT* [14], [15]. In general, the process is extremely time-consuming. However, the ATPG SAT instances have the above-mentioned special properties. A general All-SAT problem can be reduced to a

problem generally known as #(2+p)-SAT [16]. Possibilities of using dedicated 2+p-SAT solvers [26] will be a topic of our further research.

There is another application of the #SAT problem. An ATPG process often needs to discover faults that are 'hard to test' and treats them specially. The 'hardness' can be formally seen as the probability that a random test vector detects the fault. The probability can be estimated, for example by Monte Carlo simulation. However, the number of solutions of the SAT instance for a fault is directly related to this probability and determines its testability.

3.1 Solution-Preserving Reductions

In order to maximally simplify the SAT instances, we tried to apply simple transformations reducing numbers of variables and clauses. As a result, we hope it will make the instances simpler to solve and less memory consuming to store them. First, we reduce the number of variables. There often appear 1-literal clauses in the ATPG SATs. Variables of these literals must be assigned to a constant value, for the SAT to be satisfiable. Repeated application of 1-literal clause elimination will identify most of constantly set variables. The rest of them are detected by fixing the variable to a constant value and solving the SAT problem, see [26]. Next, we reduce the number of clauses by removing duplicities, absorbed clauses, and by creating resolution terms [11]. All these transformations preserve all SAT solutions. Experimental results show that in the average 57% of variables and 62% of clauses can be removed by this way. After these reductions, the terms/variables ratio sinks down to 1.67 (from 2.37), which in still in the unsatisfiability region of random SATs. The percentage of 2-literal clauses grows up to 86% (from 70%), whereas the percentage of 3-literal clauses sinks to 11% (from 24%). Notice that the reduced SAT instances are even simpler to be solved than the unreduced ones, in terms of both the clauses/variables ratio and the percentage of 3-literal clauses.

The secondary outcome of the reduction was determination of the *backbone* size [12]. The backbone is a set of variables that are fixed to a constant value for every SAT solution. The backbone is removed by the reduction, while its average size was that 57% of variables. For the randomly generated 'clones' of the benchmarks, the backbone size was 77% of variables. This allows us to state that the ATPG SAT instances are much less constrained than random ones and therefore their satisfiability is higher [12]. To prove this theory, we have generated random clones of the *reduced* instances. Thus instances, where *no backbone* is present. As a result, only 54% of the random instances were satisfiable. This clearly disproves our theory; the reason for the unsatisfiability of the random instances *is not* the backbone size.

4 Conclusions & Future Work

We have made an exploration of the nature of SAT instances generated by SAT-based ATPG tools. We have experimentally evaluated parameters of such SAT

instances and found them similar to 2+p-SAT problems, where $p = 2.37$. This makes these problems very easy to be solved by state-of-the-art SAT solvers. This fully justifies already known facts. However, the 'easiness' of ATPG SAT was formerly proven by analyzing the circuits, not the actual SAT instances produced.

Next, we have found that ATPG SAT instances significantly differ from randomly generated ones of equal parameters, particularly in terms of their satisfiability. ATPG SAT instances are mostly satisfiable, even though random instances of the same parameters should not be. We have proposed, and consequently disproved some theories on the reasons in this paper. The investigation of the satisfiability phenomenon is still in progress.

Next, we have briefly described the very motivation for these experiments - an application to a novel SAT ATPG-based test compression method. In contrast to standard SAT-based ATPGs, more than one or all SAT solutions are required. If the nature of the ATPG SATs was properly understood, problem specific #SAT-solvers might be used here. This will be a task of our further investigation.

Acknowledgement

This research has been supported by MŠMT under research program MSM6840770014 and by the grant of the Czech Grant Agency GA102/09/1668.

References

1. Agarwal, V.K., Kime, C.R., Saluja, K.K.: A tutorial on BIST, part 1: Principles. IEEE Design & Test of Computers, vol. 10, No.1 March 1993, pp.73-83, part 2: Applications, No.2, June 1993, pp. 69-77
2. Novák, O., Zahrádka, J.: COMPAS - Compressed Test Pattern Sequencer for Scan Based Circuits. Proc. of EDCC 2005, pp. 403-414
3. Rajski, J., Tyszer, J., Kassab, M., Mukherjee, N.: Embedded deterministic test. IEEE Trans. CAD, vol. 23, pp. 776-792, May 2004
4. Larrabee, T.: Test Pattern Generation Using Boolean Satisfiability. IEEE Transactions on Computer-Aided Design, pp. 4-15, Jan 1992
5. Drechsler, R., Eggersgluss, S., Fey, G., Tille, D.: Test Pattern Generation using Boolean Proof Engines. Publisher Springer Netherlands, ISBN 978-90-481-2360-5, 2009, XII, p. 192
6. Shi, J., Fey, G., Drechsler, R., Glowatz, A., Hapke, F, and Schloeffel, J.: PASSAT: Efficient SAT-based test pattern generation for industrial circuits. In IEEE Annual Symposium on VLSI (ISVLSI), pp. 212-217, 2005
7. Cook, S.A.: The complexity of theorem-proving procedures. In Proc. of the 3rd annual ACM symposium on Theory of computing, pp. 151-158, New York, NY, USA, 1971. ACM Press
8. Prasad, M. R., Chong, P., Keutzer, K.: Why is ATPG easy?. In Proc. of the 36th Annual ACM/IEEE Design Automation Conference, New Orleans, USA, June 21 - 25, 1999, pp. 22-28

9. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., and Malik, S.: Chaff: Engineering an efficient SAT solver. In Design Automation Conference (DAC), pp. 530-535, 2001
10. Selman, B.: Stochastic search and phase transitions: AI meets physics. Proc. of the 14th International Joint Conference on Artificial Intelligence, pp. 998-1002, Montreal, Que., Canada, 1995
11. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM (JACM), Vol. 7, Issue 3, July 1960, pp. 201-215
12. Monasson, R., Zecchina, R., Kirkpatrick, S. Selman, B., and Troyansky, L.: 2+p-SAT: relation of typical-case complexity to the nature of the phase transition. In Random Structures & Algorithms, Vol. 15, Issue 3-4, Oct.-Dec. 1999, pp. 414 - 435
13. Balcárek, J.: Test Patterns Compression Techniques Based on SAT Solving for Scan-Based Digital Circuits. Počítačové architektury&diagnostika, Soláň (CR), 9.-11.9. 2009, pp. 26-31
14. Sinz, C., Biere, A.: Extended Resolution Proofs for Conjoining BDDs. In Proc. 1st Intl. Computer Science Symp. in Russia (CSR 2006), St. Petersburg, Russia, Lecture Notes in Computer Science (LNCS), vol. 3967, Springer 2006
15. Jin, H., Han, H., Somenzi, F.: Efficient Conflict Analysis for Finding All Satisfying Assignments of a Boolean Circuit, Lecture notes in computer science , vol. 3440/2005, pp. 287-300
16. Kutzkov, K.: New upper bound for the #3-SAT problem. In: Information Processing Letters 105 (2007)
17. Aspvall, B., Plass, M.F., Tarjan, R.E.: A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. Inf. Process. Lett. 8(3): 121-123 (1979)
18. Klenke, R. H., R. D. Williams, Aylor, J. H.: Parallel-Processing Techniques for Automatic Test Pattern Generation. IEEE Computer, January 1992, pp. 71-84.
19. Goel, P.: An implicit enumeration algorithm to generate tests for combinational logic. IEEE Transactions on Computers, 30:215222, 1981.
20. Fujiwara H., Shimono, T.: On the acceleration of test generation algorithms. IEEE Transactions on Computers, 32:1137-1144, 1983.
21. Eén, N., Sorensson, N.: An Extensible SAT-solver. Lecture Notes in Computer Science, Theory and Applications of Satisfiability Testing, vol. 2919/2004, pp. 333-336, 2004
22. Brglez, F., Fujiwara, H.: A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan. Proc. of International Symposium on Circuits and Systems, pp. 663-698, 1985
23. Brglez, F., Bryan, D., Kozminski, K.: Combinational Profiles of Sequential Benchmark Circuits. Proc. of International Symposium of Circuits and Systems, pp. 1929-1934, 1989
24. Corno, F., Sonza Reorda, M. Squillero, G.: RT-Level ITC 99 Benchmarks and First ATPG Results. IEEE Design & Test of Computers, July-August 2000, pp. 44-53
25. Su, C., and Hwang, K.: A Serial Scan Test Vector Compression Methodology. Proc. ITC 1993, pp. 981-988
26. Singer, J., Gent, I.P., Smaill, A.: Local Search on Random 2+p-SAT. In Proc. of the 14th ECAI, pp. 113-117, 2000
27. Kravchuk, O., Pullan, W., Thornton, J., Sattar, A.: An investigation of variable relationships in 3-SAT problems. Al 2002: Advances In Artificial Intelligence, 2557, 2002, pp. 579-590