Structural Heuristics for SAT-based ATPG

Daniel Tille Stephan Eggersglüß Hoang M. Le Rolf Drechsler
Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
{tille,segg,hle,drechsle}@informatik.uni-bremen.de

***Abstract*—Due to ever increasing design sizes more efficient tools for *Automatic Test Pattern Generation* (ATPG) are needed. The application of the Boolean satisfiability problem (SAT) to ATPG has been shown to be a robust alternative to traditional ATPG techniques. A major challenge of research in the field of SAT-based ATPG is to obtain a robust algorithm which can solve hard SAT instances reliably without slowing down easy-to-solve SAT instances. This is particular important, since easy-to-solve SAT instances form the majority of an ATPG run. This paper proposes two structural heuristics. The first one uses testability measurements to obtain an improved initial variable order, while the second heuristic prunes many easy-to-test faults by finding easy-to-control paths. Experimental results on large industrial designs confirm that the proposed methodologies result in a significant overall speed-up.**

I. Introduction

According to Moore's law, the size of integrated circuits doubles every 18 months. This continuous growth requires a constant improvement of state-of-the-art *Electronic Design Automation* (EDA) tools. The post-production test is a vital step in the design flow. It ensures the functional correctness of the produced chips. This step is very important to guarantee high quality.

In practice, a fault model is usually used to abstract from the physical defects. To test the circuit for correctness with respect to the applied fault model, test patterns have to be computed. In this work, the stuck-at fault model [6] is used. To generate a test pattern for a stuck-at fault, there exist many sophisticated algorithms. If there is a test pattern for a particular fault F, then F is called *testable*; otherwise F is called *untestable*.

The D-algorithm [29] was the first algorithm that traversed the search space by backtracking. Improvements concerning decision strategies as well as propagation and justification were given in PODEM [18] and FAN [16]. Further algorithms are Socrates [30] and Hannibal [23]. All these algorithms have in common that they directly work on the circuit structure.

In contrast, there also exist approaches based on Boolean satisfiability [24], [31], [26], [32], [17], [11]. Due to the lack of efficient SAT solvers, the early approaches did not become popular. The improvements made for solving SAT in the last decade [27], [28], [19], [15], [4], however, resulted in a renewed interest in this topic. Recent research work shows that SAT-based ATPG can be successfully applied to large industrial circuits [11].

SAT-based ATPG consists of two separate stages: first a SAT instance is generated and second this instance is solved. In contrast to, e.g. SAT-based verification, a large number of instances has to be solved. Most of them are easily solvable, however, there also exist very huge and very hard ones. A major challenge of SAT-based ATPG research is thus to obtain a robust framework in which hard instances can be solved reliably without slowing down the classification of easy-to-test faults.

This paper proposes two structural heuristics that aim for speeding up the entire ATPG process and enhancing the robustness. The first heuristic incorporates testability measurements used for decision making in traditional ATPG. These measurements are applied to obtain an improved initial variable order for the decision heuristic of the SAT solver. This, in particular, accelerates the search for a test pattern for hard-to-test faults. However, a performance improvement for easy-to-test faults cannot be observed. As a disadvantage, overhead is caused by the decreasing diversity of the generated test patterns. To overcome this drawback, a metric is introduced to determine dynamically which order should be used. By this, the advantages of both, the default initial variable order and the proposed order are exploited.

The robustness of the ATPG process is further increased by the second technique. A structural heuristic is applied in a preprocessing step (prior to the SAT instance generation) to find easy-to-control paths quickly and such prunes many easy-to-test faults. In this approach, testability measurements are used to guide the heuristic.

The rest of this paper is organized as follows. The next section gives a brief introduction into state-of-the-art SAT-based ATPG algorithms. Section III presents the basic idea and introduces testability measurements. The structural heuristics are proposed and discussed in Section IV. Experimental results are given in Section V and finally the paper is concluded in Section VI.

II. Related Work

This section gives a brief overview on state-of-the-art SAT-based ATPG. To get further insight into this topic, see e.g. [12]. For a general overview on the ATPG problem and the classical algorithms to solve it, we refer to e.g. [7].

A. SAT-based ATPG

To compute a test pattern for a stuck-at fault F, an assignment to the inputs has to be found that guarantees at least one different output value at the correct circuit and the faulty circuit. Classical algorithms work directly on a circuit structure to either obtain a test pattern or to prove that no test pattern exists.

The ATPG problem, however, can also be seen as a the following decision problem. "Given a circuit C and a fault F. Is there a test pattern that makes F observable?" This problem can be transformed into a Boolean satisfiability problem, i.e. a Boolean formula.[1] This formula is satisfiable if, and only if, the fault is testable. A test pattern – if it exists – can be derived directly from the satisfying assignment of those variables corresponding to the primary inputs. Any SAT solver can be used to solve the problem.

[1]The ATPG problem is – as well as the SAT problem – NP-complete.

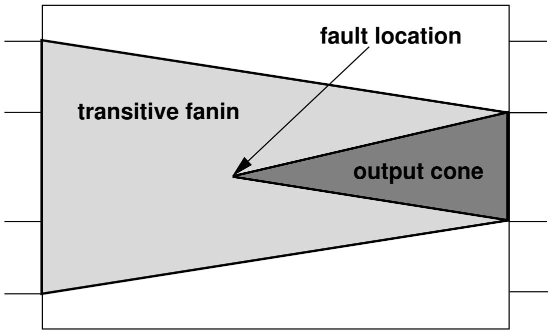


Fig. 1: Extraction of the influenced circuit parts

Modern SAT solvers work on instances given in *Conjunctive Normal Form* (CNF). A CNF is a conjunction of clauses, a clause is a disjunction of literals and a literal is a positive or negative occurrence of a Boolean variable. A CNF is satisfied if all clauses are satisfied; a clause is satisfied if at least one of its literals is satisfied; a positive or negative literal is satisfied if the variable is assigned positively or negatively, respectively.

Although the use of SAT to effectively solve circuit-oriented problems has been studied in detail in the last years, SAT-based ATPG is very special. SAT-based approaches for, e.g., equivalence checking [21] or model checking [3] usually have to solve only one or few difficult problem instances. One SAT-based ATPG run, however, contains thousands of SAT instances. Most of them can be solved quite easily within a runtime of only a few CPU milliseconds. However, there also occur hard-to-solve instances requiring several minutes of CPU time. Furthermore, the CNF sizes can vary significantly. Instances with less than one hundred variables, on the one hand, and more than one million variables, on the other hand, can occur for the same benchmark circuit. Thus, a SAT-based ATPG framework has to be very robust in terms of solving SAT instances. For example, preprocessing methods as proposed in [14] are not feasible for SAT-based ATPG. The overhead on small, easy-to-solve instances would be too large.

Another interesting fact is that in SAT-based ATPG, unsatisfiability, i.e. untestability, can often be proven much more easily than satisfiability, i.e. testability [34]. This can be explained as follows. The reason for untestability is often due to redundancy in the circuit, located close to the fault site, or due to restrictions to the primary inputs. In both cases, the SAT solver may be able to find an unsatisfiable core [20], [38] just by propagation steps, i.e. no decision has to be made. If untestability cannot be proven this immediately, the reason will most likely be found very soon anyway due to the inherent powerful conflict analysis. Incorporating SAT techniques to accelerate proving unsatisfiability, e.g. as described in [36], where unobservability constraints are encoded in the SAT instance, cannot be applied beneficially in SAT-based ATPG.

Finding a test pattern, however, is often more complex [33]. This is due to the stopping criterion of modern SAT solvers. A SAT instance is proven to be satisfiable if, and only if, all variables are assigned and no conflict occurred. Assigning all variables can be a very time-consuming step, especially for circuit parts containing many reconvergences.

B. Circuit-to-CNF Conversion

Consider the schematically depicted circuit in Figure 1. Here, a brief overview on the circuit-to-CNF conversion is given.

After the fault location has been marked, the fault site's output cone is traversed by a depth first search. This determines all *Primary Outputs* (POs) that are structurally influenced by the fault, i.e. all POs where a difference between the faulty circuit and the faultless circuit could be observed. The transitive fanin of these POs influences the detection of the fault and must be marked, too. To generate the SAT instance for the given fault, this part of the circuit has to be considered.

As introduced in [31], two Boolean variables g_c and g_f are assigned to each gate g in order to represent the gate's value in the correct circuit and in the faulty circuit, respectively.[2] A gate's CNF is generated by building its characteristic function. The conjunction of all CNFs results in the CNF for the circuit [35].

To find a difference between the correct circuit and the faulty circuit, an additional Boolean variable g_d is assigned to each gate. If the variable g_d is true, the gate's values in both circuits differ. Therefore, the constraint

$$g_d = 1 \rightarrow g_c \neq g_f$$

is added to the CNF in form of the two clauses

$$(\overline{g}_d + g_c + g_f) \cdot (\overline{g}_d + \overline{g}_c + \overline{g}_f).$$

To compute a test pattern for a fault, there must be a path from the fault site to an output, where the assignment of each variable g_d is true. Following the notation in [31], this path is called a *D-chain*. Therefore, if a gate is on a D-chain, one successor must be on a D-chain as well. This property – encoded by the constraint

$$g_d \rightarrow \bigvee_{i=1}^{n} h_d^i,$$

where the gates $h^1, \ldots, h^n$ denote the successors of gate g – is also added to the CNF. Moreover, the variable g_d^f, where the gate g^f represents the faulty gate, is set to true in order to inject a difference at the fault site.

As a result, the SAT instance generated this way is satisfiable if, and only if, a D-chain exists, i.e. the SAT instance is satisfiable if, and only if, the fault is testable. The same result could be achieved by using a miter circuit [5]. However, this construction has an important advantage. Since the correct and the faulty circuit are connected between each gate (and not only between the outputs as done in a miter circuit), unsatisfiability, i.e. untestability, can often be proven immediately (as explained above).

Finally, this SAT instance is given to a SAT solver. After the classification, the CNF is completely discarded. Therefore, the circuit-to-CNF conversion has to be done for each single target fault and consumes a large part of the overall runtime as already stated in [26], [33].

As explained above, unlike in other circuit-oriented problems, thousands of SAT instances have to be generated in SAT-based ATPG. It has been shown in [26], [33] that the time needed to generate an instance often exceeds the time needed to solve it. The instance generation step would be unnecessary if a circuit-based SAT solver [25], [37] is used. However, due to our experimental studies, the application of circuit-SAT solvers to the ATPG problem results in increased overall runtime.

[2]Since the values of both circuits may differ only in the fault site's fanout cone, the variable g_f is assigned only to gates g in this cone of influence.

TABLE I: Computation of SCOAP controllability values

Gate type	$contr\_0$	$contr\_1$
PI	1	1
AND	$\min_{h_i}(h_i.contr\_0)+1$	$\sum_{h_i} h_i.contr\_1+1$
OR	$\sum_{h_i} h_i.contr\_0+1$	$\min_{h_i}(h_i.contr\_1)+1$
NOT	$h_0.contr\_1$	$h_0.contr\_0$

III. Basic Idea

After a CNF has been generated, an arbitrary SAT solver is used to solve it. State-of-the-art SAT solvers base on the DPLL algorithm [10], [9] and use dynamic variable ordering heuristics [28]. These heuristics prefer variables that occurred recently in conflicts. Due to conflict-based learning [27], the search space can thus be pruned by running into conflicts quickly.

The initial variable order, however, is quite straightforward: either each variable gets the same weight or the weight correlates with the total number of occurrences in the CNF. The SAT solver may branch on unsuitable variables during the beginning of the solving step. Algorithms like MINCE [1], FORCE [2] or ACCORD [13] overcome this drawback. Those techniques try to find an optimal (static) variable order for solving SAT (or building BDDs) by capturing structural properties of Boolean functions arising from real-world applications. The connectivity of variables is calculated by examining the CNF structure. Those approaches, however, focus on solving one hard-to-solve instance faster. For SAT-based ATPG, where most instances are easy to solve they produce considerable overhead. Therefore those approaches are unsuitable for SAT-based ATPG.

The underlying problem in SAT-based ATPG is circuit-based. Traditional ATPG algorithms like PODEM [18] or FAN [16] use testability measurements, e.g. SCOAP [22], when a decision has to be made. Those values can be calculated in linear time of the number of gates during a preprocess prior to the actual ATPG run. Testability measurements indicate how difficult it is to control or to observe a signal, i.e. how difficult it is to set a particular signal to 0 or 1.

The SCOAP testability measurement consists of three different values:

- 0-controllability – This value determines the difficulty to justify a 0 value on this signal line.
- 1-controllability – Complementary to 0-controllability, this value determines the difficulty to justify a 1 value on this signal line.
- observability – This value determines the difficulty to propagate the gate's value to an output.

In this work, only the controllability measurements are considered. They are denoted by $contr\_0$ and $contr\_1$ in the following. Table I gives an overview on how to compute those values for a basic gate g with n predecessors. The predecessors of g are denoted by h_i with $0 \leq i \leq n-1$.

The following example shows the use of controllability values in order to guide traditional ATPG engines.

Example 1: Consider the circuit depicted in Figure 2. Each line's controllability values are denoted as ($contr\_0$,$contr\_1$).

Assume, line h must be set to the value 1. To justify this value, one of the NAND gate's inputs has to be set to 0. At this point, it has be decided which input is used for the justification. Looking on the circuit structure, it can be seen that line a would be the better choice since it is a primary input and can be set to 0 immediately. The justification with

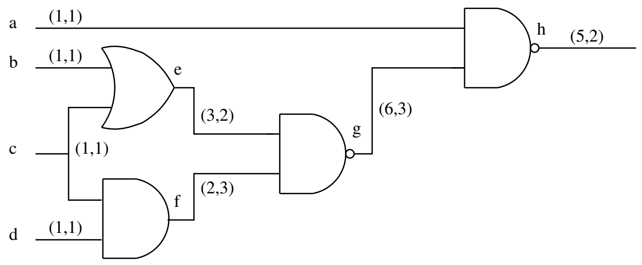


Fig. 2: Example for SCOAP controllability values

line g would be more complex. This situation is also reflected by the controllability values. The smaller the controllability value is the easier can the signal value be justified.

In SAT-based ATPG, testability measurements are not used so far. An approach to use them in order to guide the SAT solver to branch on easy-to-control variables is proposed in the following.

IV. Structural Heuristics

In this section, the main contribution of this paper is given.

A. Initial Order and Polarity Heuristic

The *Initial Order and Polarity Heuristic* (IOPH) uses the SCOAP controllability values described above to make sure that the SAT solver accesses easy-to-control regions first. This is accomplished by changing the default initial variable order. The initial variable order depends on the *activity value*[3] of the variables. In this approach the initial activity value is calculated as follows. Let k be a large constant number, e.g. the highest SCOAP value assigned to a gate. Then, the activity value $g.a$ for some gate g is computed by

$$g.a = k - \min(contr\_0, contr\_1)$$

As a result, the variables associated with a low controllability value are initially preferred by the decision heuristic. Variables corresponding to easy-to-control gates are thus picked by the decision heuristic first. The polarity of this variable is considered as well. It is assigned to the value, that is easier to control. This approach avoids to enter hard-to-control regions of the circuit at the beginning and increases the likelihood that a test pattern is found quickly within an easy-to-control circuit part. Since unsatisfiable instances are often easy to classify, the heuristic does not slow down the solving time for those faults. Because the IOPH only changes the initial variable order, the influence of the controllability values decreases during the solving process.

The following simple example illustrates the derivation of the initial order.

Example 2: Again consider the circuit depicted in Figure 2.

The controllability values presented at the signal lines are used to determine an initial variable order. Here, it is given by

$$a=0, b=0, c=0, d=0, e=1, f=0, h=1, g=1$$

where $x = v$ means the variable correlated to the gate x is set to the Boolean value v. As can be seen, this initial variable order makes the SAT solver branch on primary inputs first and continues with variables contained in easy-to-control regions.

[3]The more often and the more recently a variable occured in a conflict, the larger is the activity value. The variable with highest activity value is chosen to branch on.

The effectiveness of the approach is shown in Figure 3. It presents four diagrams depicting the average SAT solving time (on the y-axes) in relation to the size of the SAT instances, i.e. the number of variables contained in the CNF (on the x-axes). It can clearly be seen that the influence of the proposed IOPH method largely depends on the number of variables. There is only a negligible difference between the two approaches on small SAT instances. On large CNFs, on the other hand, there is often a wide gap in the average runtimes for both techniques.

However, the IOPH technique contains a considerable drawback compared to the default heuristic. Since the testability measurements are computed once for the entire circuit, the initial variable orders of SAT instances for different faults are similar. Therefore, the derived test patterns are often very similar as well. The *fault simulator*[4] performs very badly on those kind of test patterns since fewer "additional" faults can be detected. The default decision heuristic, however, produces many "random" patterns that cover, i.e. detect, a large portion of the fault list. Even if the average runtime for each fault remains equal, the overall runtime of the ATPG run can slow down due to the increased number of faults which have to be targeted.

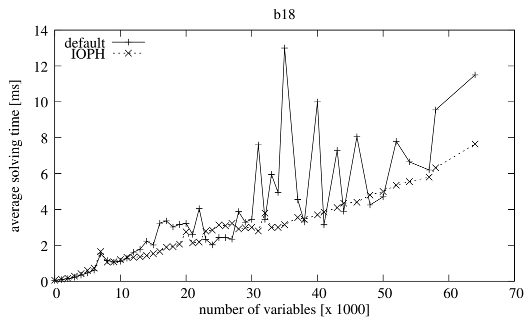
Based on those observations, an improvement that combines the strengths of the two initial variable orders is proposed. For this improvement, a new metric is introduced. Up to a certain limit on the number of variables, instances will always be solved using the default initial order. The limit should be high enough to assure the diversity of the patterns. Beyond that limit, instances will be divided into classes based on the number of variables they contain. For each of those classes the heuristic predicts which initial order is better to use. That is done by alternately applying each initial order on the first N encountered instances of each class. The required runtime is stored. Note, only "non-trivial" instances, i.e. instances that cannot be solved within 2 restarts of the SAT solver, are taken into account for this metric. The heuristic with the lower total time spent on SAT solving will be selected for following instances contained in the class.

The current implementation uses the following parameters. The lower limit for the number of variables is 20,000, i.e. CNFs containing less than 20,000 variables are usually easy to solve and do not require a special treatment of the initial order. This is confirmed by Figure 3. Furthermore, $N = 10$ and the range for the classes the CNFs are divided into is 1,000 variables. This improvement to the IOPH technique is denoted by *IOPH+*. The experiments of ATPG runs incorporating these techniques are shown in Section V.

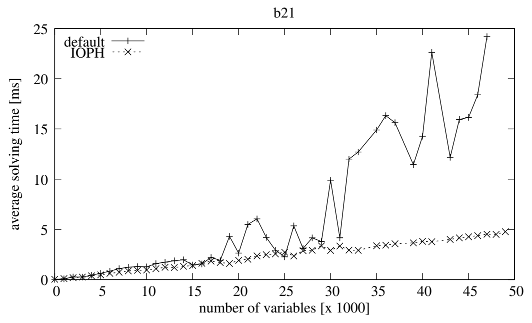
B. Test Pattern Generating Heuristic

As mentioned in Section II, the time needed to generate a SAT instance is a significant part of the overall runtime and often even dominates it. Moreover, testable faults, i.e. satisfiable SAT instances, are often harder to solve than untestable faults, i.e. unsatisfiable SAT instances. Therefore, it would be beneficial to classify a fault without generating a CNF at all. This section proposes a preprocessing heuristic that is able to find a test pattern prior to the SAT instance generation by enabling easy-to-control paths.

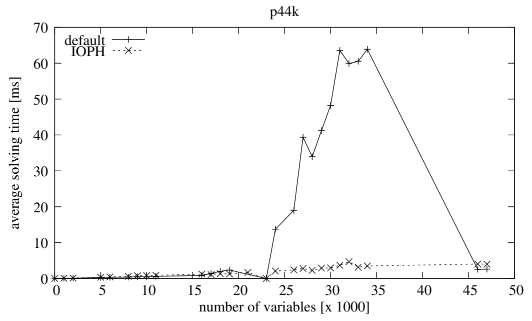
[4]After computing a test pattern, a fault simulator determines all faults that can be detected by this particular test pattern as well. Those faults are discarded from the fault list and do not need to be considered during the subsequent ATPG.



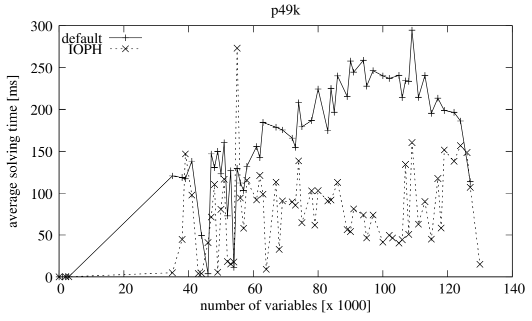
(a)



(b)



(c)



(d)

Fig. 3: Dependency between average solving time and number of variables

As explained in Section II-B, the circuit-to-CNF conversion is a graph algorithm working on the circuit structure. This procedure can be modified in order to check for testability without generating a CNF. This is accomplished by emulating a part of the D-algorithm [29] – a classical ATPG method. This algorithm works on the 4-valued logic

$$\{0, 1, D, \bar{D}\}$$

where D denotes a difference between the value in the faulty circuit (0) and the value in the correct circuit (1). The symbol

TABLE II: Experimental results

			Default				IOPH				IOPH+				IOPH+ & TPGH		
Circuit	Targets	Untest.	CF	∅SAT t.	Ab.	Time	CF	∅SAT t.	Ab.	Time	CF	∅SAT t.	Ab.	Time	Ab.	Time	Suc. rate
b17	76,493	1,958	11,935	0.33ms	0	4:21m	15,309	0.62ms	0	6:43m	11,885	0.37ms	0	4:21m	0	3:48m	19.91%
b18	264,043	2,844	30,530	0.49ms	0	14:10m	39,621	0.59ms	0	22:41m	30,770	0.49ms	0	14:00m	0	10:51m	17.03%
b20	45,461	319	7,463	0.74ms	0	3:37m	13,465	0.51ms	0	5:53m	7,489	0.73ms	0	3:31m	0	1:55m	17.41%
b21	46,156	378	7,708	0.74ms	0	3:46m	13,885	0.50ms	0	6:14m	7,727	0.73ms	0	3:39m	0	1:59m	16.55%
b22	67,540	344	9,178	0.72ms	0	4:34m	16,310	0.53ms	0	7:48m	9,152	0.71ms	0	4:24m	0	2:29m	18.22%
p44k	64,105	2,385	19,566	22.88ms	0	1:40h	22,081	2.11ms	0	38:28m	20,710	2.38ms	0	35:19m	0	10:34m	66.59%
p49k	142,461	1,296	8,410	176.68ms	2,535	5:01h	4,658	92.22ms	265	1:45h	5,345	111.45ms	602	2:16h	547	1:30h	27.58%
p77k	163,310	9,181	10,347	0.03ms	0	0:22m	10,658	0.01ms	0	0:22m	10,347	0.02ms	0	0:21m	0	0:14m	7.22%
p80k	197,834	124	11,223	2.7ms	0	7:30m	26,932	1.21ms	0	11:43m	11,197	2.53ms	0	7:02m	0	2:16m	73.66%
p88k	147,742	2,640	21,638	0.14ms	0	3:31m	26,236	0.15ms	0	4:40m	21,638	0.14ms	0	3:26m	0	2:26m	54.55%
p99k	162,019	2,141	11,892	0.26ms	1	2:09m	13,380	0.38ms	1	2:46m	11,892	0.26ms	1	2:06m	1	1:28m	38.02%
p177k	268,176	13,840	27,955	27.84ms	0	3:04h	35,897	7.74ms	0	2:08h	29,760	6.91ms	0	1:29h	0	46:39m	25.02%
p456k	740,660	35,396	60,545	3.07ms	170	1:00h	65,526	1.62ms	91	51:52m	60,622	2.47ms	117	53:46m	143	48:43m	16.95%
p462k	673,465	132,249	155,910	2.09ms	13	1:35h	159,962	2.23ms	23	1:43h	156,590	2.04ms	14	1:32h	17	1:16h	10.36%
p565k	1,025,273	28,287	48,734	0.07ms	0	9:56m	49,988	0.07ms	0	10:20m	48,734	0.07ms	0	9:48m	0	9:04m	25.81%
p1330k	1,510,574	44,299	71,821	0.44ms	0	1:26h	74,727	0.38ms	0	1:30h	71,821	0.43ms	0	1:22h	0	1:10h	27.22%
p2787k	2,395,388	651,868	701,287	4.24ms	1,877	23:10h	708,708	2.23ms	1,050	19:27h	701,506	2.55ms	1,011	19:05h	1,357	19:10h	5.20%
p3327k	4,557,842	109,622	349,822	33.77ms	2,053	47:13h	366,262	40.13ms	525	54:49h	353,531	19.57ms	726	33:03h	591	9:31h	44.65%
p3852k	5,507,779	164,988	387,060	9.87ms	1,637	21:03h	414,180	2.78ms	374	14:41h	391,212	3.21ms	445	14:00h	343	7:14h	36.76%

$\bar{D}$ denotes the negation of D.

The proposed approach works as follows. First, the connection containing the fault site is set to the faulty value followed by the attempt to justify this value by backward propagation. Second, the fault effect, i.e. the difference between both values, is tried to be propagated to each single output. This is also done by backward propagation.

Whenever a decision has to be made, i.e. it has to be decided which gate input is used to justify the current value, the easiest possibility is chosen. This is done by using the SCOAP testability measurement. If it is possible to compute a test pattern this way, this is done very quickly, since an easy-to-control path is used. However, if it is not possible to prove testability on such a path, the preprocess is stopped and the ordinary SAT instance generation is started.

Note, since this is no backtracking algorithm, this propagation approach is not complete, i.e. a solution cannot be guaranteed. Untestable faults cannot be classified at all using this technique. However, as will be confirmed by the experimental results presented in the next section, many faults can be pruned this way. This technique is denoted by TPGH.

V. Experimental Results

Experimental results are given in the following. The heuristics presented in Section IV were implemented as a prototype into an ATPG framework. MiniSat v1.14 [15] was used to solve the SAT instances.[5] Two benchmark sets have been considered: the publicly available ITC'99 benchmarks [8] and industrial circuits provided by NXP Semiconductors Germany GmbH, Hamburg, Germany. The names of the NXP benchmarks indicate the number of gates roughly contained in a circuit, e.g. the circuit p3852k consists of approximately 3.85 million gates.

All experiments (except for p3327k and p3852k) were carried out on an AMD Athlon 64 3500+ (2.2 GHz, 3 GByte, Linux). The two largest benchmarks, however, need more main memory to run and thus were conducted on a Dual Dual-Core AMD Opteron 2220 (2.8 GHz, 32 GByte, Linux). The fault classification of one target fault is aborted after 7 MiniSat restarts. This corresponds to 3,221 conflicts during the solving step.

[5]We also tried to use MiniSat v2.0, but got permanently worse results. This can be explained by the "binary clause trick" contained in MiniSat v1.14, but not in MiniSat v2.0.

The effectiveness of the proposed techniques is shown in Table II. The first column presents the circuit's name. The number of target faults, i.e. the number of faults that have to be considered after discarding equivalent faults (known as *fault collapsing*) is given in the second column. The third column shows the number of untestable faults in the circuit. To evaluate the different methodologies, the number of considered faults during ATPG (column *CF*), the average time needed to solve one SAT instance (column *∅SAT t.*), the number of aborted fault classifications (column *Ab.*) and the total runtime of the ATPG process (column *Time*) are given.

The experimental results using the default initial variable order are presented in column *Default*. Comparing those with the results incorporating the IOPH method – given in column *IOPH* – it can be seen that the number of aborted fault classifications can be reduced considerably. Furthermore, the overall runtime is significantly influenced. On some benchmarks, there is a slowdown observable, e.g. b18, p80k and p3327k, whereas a significant acceleration was achieved for example on circuits p44k, p49k and p3852k. Even if the average solving time of one SAT instance can be decreased, the overall solving time may slow down. As explained above, this is due to the significantly increased number of considered faults. This motivates the use of the proposed metric resulting in the IOPH+ technique that is discussed in the following.

It can be seen that using the IOPH+ method (column *IOPH+*) results on the one hand in a slightly increased number of aborted faults compared to the IOPH method. Compared to the default initial variable ordering, on the other hand, a significant reduction can still be observed. However, the use of the IOPH+ technique consistently results in better runtimes. On some circuits, there is only a small acceleration. On the other hand, speed-ups of a factor of 2.8 and 2.2 can be achieved for the circuits p44k and p49k, respectively.

The number of aborts changes slightly when adding the TPGH technique to the IOPH+ approach (column *IOPH+ & TPGH*). The runtime, however, can largely be improved using the IOPH+ & TPGH approach. Except for circuit p2787k, where a small slowdown appears, all ATPG runs can be performed with decreased runtime. A considerable acceleration can be observed on most benchmarks. On circuit p44k, a remarkable speed-up of a factor of 9.5 can be achieved.

The experimental results of the latter approach also contain an overview on the success rate of the TPGH method which is presented in column *Suc. rate*. It gives the relative number of successful classifications using the TPGH method with respect to all considered faults. The lowest rate can be observed at circuit p2787k. This can be explained by the extreme low fault coverage, i.e. the unusual large number of untestable faults. The TPGH approach, however, aims for accelerating the classification of testable faults. Considering circuit p44k, 2 out of 3 faults can be classified using the TPGH technique; on circuit p80k, this number is even larger. Here, almost 3 out of 4 fault can be classified using the preprocessing method.

To summarize, the proposed structural heuristics are able to increase the robustness of a SAT-based ATPG framework.

VI. Conclusions

The paper presents two structural heuristics in order to speed up SAT-based ATPG and to improve the overall robustness. The first technique incorporates testability measurements – known from traditional ATPG algorithms – to calculate an initial variable order for the SAT solver's decision heuristic. The new initial variable order improves the performance especially for large SAT instances. However, for easy-to-test faults, no considerable speed-up can be observed. As a side effect, the number of target faults grows due to the decreasing diversity of test patterns. Therefore, a metric is introduced to dynamically decide which variable order should be applied for each target fault. As a result, the advantages of both variable orders can be exploited and the overall robustness is increased.

The second method accelerates the fault classification by finding a test pattern on the circuit structure through easy-to-control paths prior to the actual SAT instance generation. By this, many easy-to-test faults are pruned. The efficiency of both approaches is confirmed by experiments conducted on large industrial circuits.

Acknowledgment

This work was funded in part by DFG grant DR 287/15-1.

References

[1] F. Aloul, I. Markov, and K. Sakallah. MINCE: A static global variable-ordering for SAT and BDD. In *Proceedings of the International Workshop on Logic & Synthesis*, pages 281–286, 2001.

[2] F. Aloul, I. Markov, and K. Sakallah. FORCE: A fast and easy-to-implement variable-ordering heuristic. In *Proceedings of the Great Lakes Symposium on VLSI*, pages 116–119, 2003.

[3] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207, 1999.

[4] A. Biere, M. Heule, H. v. Maaren, and T. Walsh. *Handbook of satisfiability*. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009.

[5] D. Brand. Redundancy and don't cares in logic synthesis. *IEEE Transactions on Computers*, 32(10):947–952, 1983.

[6] M. A. Breuer and A. D. Friedman. *Diagnosis & reliable design of digital systems*. Computer Science Press, 1976.

[7] M. L. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic, 2000.

[8] F. Corno, M. Sonza-Reorda, and G. Squillero. RT-level ITC 99 benchmarks and first ATPG results. In *Proceedings of the IEEE Design & Test of Computers*, pages 44–53, 2000.

[9] M. Davis, G. Logeman, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

[10] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.

[11] R. Drechsler, S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille. On acceleration of SAT-based ATPG for industrial designs. *IEEE Transactions on Computer-Aided Design for Circuits and Systems*, 27(7):1329–1333, 2008.

[12] R. Drechsler, S. Eggersglüß, G. Fey, and D. Tille. *Test Pattern Generation using Boolean Proof Engines*. Springer, 2009.

[13] V. Durairaj and P. Kalla. Variable ordering for efficient SAT search by analyzing constraint-variable dependencies. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, volume 3569 of *Lecture Notes in Computer Science*, pages 415–422, 2005.

[14] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75, 2005.

[15] N. Eén and N. Sörensson. An extensible SAT solver. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518, 2004.

[16] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, 32(12):1137–1144, 1983.

[17] E. Gizdarski and H. Fujiwara. SPIRIT: A highly robust combinational test generation algorithm. *IEEE Transactions on Computer-Aided Design for Circuits and Systems*, 21(12):1446–1458, 2002.

[18] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic. *IEEE Transactions on Computers*, 30(3):215–222, 1981.

[19] E. Goldberg and Y. Novikov. BerkMin: a fast and robust SAT-solver. In *Proceedings of Design, Automation and Test in Europe*, pages 142–149, 2002.

[20] E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Proceedings of Design, Automation and Test in Europe*, pages 886–891, 2003.

[21] E. I. Goldberg, M. R. Prasad, and R. K. Brayton. Using SAT for combinational equivalence checking. In *Proceedings of Design, Automation and Test in Europe*, pages 114–121, 2001.

[22] L. H. Goldstein and E. L. Thigpen. SCOAP: Sandia controllability/observability analysis program. In *Proceedings of the Design Automation Conference*, pages 190–196, 1980.

[23] W. Kunz. HANNIBAL: An efficient tool for logic verification based on recursive learning. In *Proceedings of the International Conference on Computer-Aided Design*, pages 538–543, 1993.

[24] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design for Circuits and Systems*, 11(1):4–15, 1992.

[25] F. Lu, L.-C. Wang, K.-T. Cheng, and R. Huang. A circuit SAT solver with signal correlation guided learning. In *Proceedings of Design, Automation and Test in Europe*, pages 892–897, 2003.

[26] J. P. Marques-Silva and K. A. Sakallah. Robust search algorithms for test pattern generation. In *Proceedings of the International Symposium on Fault-Tolerant Computing*, pages 152–157, 1997.

[27] J. P. Marques-Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[28] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*, pages 530–535, 2001.

[29] J. P. Roth. Diagnosis of automata failures: A calculus and a method. *IBM Journal of Research and Development*, 10:278–281, 1966.

[30] M. H. Schulz, E. Trischler, and T. M. Sarfert. SOCRATES: A highly efficient automatic test pattern generation system. *IEEE Transactions on Computer-Aided Design for Circuits and Systems*, 7(1):126–137, 1988.

[31] P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Transactions on Computer-Aided Design for Circuits and Systems*, 15:1167–1176, 1996.

[32] P. Tafertshofer, A. Ganz, and K. J. Antreich. IGRAINE - an implication graph based engine for fast implication, justification, and propagation. *IEEE Transactions on Computer-Aided Design for Circuits and Systems*, 19(8):907–927, 2000.

[33] D. Tille and R. Drechsler. Incremental SAT-instance generation for SAT-based ATPG. In *Proceedings of the IEEE Workshop on Design and Diagnosis of Electronic Circuits and Systems*, pages 68–73, 2008.

[34] D. Tille and R. Drechsler. A fast untestability proof for SAT-based ATPG. In *Proceedings of the IEEE Symposium on Design and Diagnosis of Electronic Circuits and Systems*, pages 38–43, 2009.

[35] G. S. Tseitin. On the complexity of derivation in the propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, Part II:115–125, 1968.

[36] M. N. Velev. Encoding global unobservability for efficient translation to SAT. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, pages 197–204, 2004.

[37] C.-A. Wu, T.-H. Lin, C.-C. Lee, and C.-Y. Huang. QuteSAT: A robust circuit-based SAT solver for complex circuit structure. In *Proceedings of Design, Automation and Test in Europe*, pages 1313–1318, 2007.

[38] L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings of Design, Automation and Test in Europe*, pages 880–885, 2003.