Accurate CEGAR-based ATPG in Presence of Unknown Values for Large Industrial Designs

Karsten Scheibler, Dominik Erb, Bernd Becker
University of Freiburg, Germany, {scheibler | erbd | becker}@informatik.uni-freiburg.de

***Abstract*—Unknown values emerge during the design and test generation process as well as during later test application and system operation. They adversely affect the test quality by reducing the controllability and observability of internal circuit structures – resulting in a loss of fault coverage. To handle unknown values, conventional test generation algorithms as used in state-of-the-art commercial tools, rely on n-valued algebras. However, n-valued algebras introduce pessimism as soon as X-values reconverge. Consequently, these algorithms fail to compute the accurate result.**

This paper focuses on a new highly incremental CEGAR-based algorithm that overcomes these limitations and hence is completely accurate in presence of unknown values. It relies on a modified SAT-solver tailored for this specific problem. The experimental results for circuits with up to 2 400 000 gates show that this combination allows high accuracy and high scalability at the same time. Compared to a state-of-the-art commercial tool, the fault coverage could be increased significantly. Furthermore, the runtime is reduced remarkably compared to a QBF-based encoding of the problem.

***Index Terms*—SAT, CEGAR, test generation, ATPG, unknown values**

I. Introduction

Unknown values (X-values) are a pain in VLSI testing. They reduce the controllability and observability of internal circuit structures resulting in an adverse impact on the test quality. Unknown values emerge during the design and test generation process as well as during later test application and system operation – e.g. by uninitialized sequential elements, clock-domain crossings or multi-cycle paths. Classically, unknown values are evaluated pessimistically for example using n-valued algebras [1, 2]. However, all these algorithms suffer from the same problem that they are unable to accurately determine the signal values as soon as X-values reconverge. This results in a reduced fault-coverage or a higher number of patterns that need to be considered. Consequently, in the last years several algorithms have been published to overcome these limitations.

Related work: In [3] the first accurate approach for ATPG in presence of unknown values was presented. This approach utilizes a QBF-solver and allowed for the first time the accurate evaluation of faults in presence of unknown values. However, the accurate analysis comes at the price of high runtimes and a large number of aborted faults.

In order to lower the runtime while keeping a high fault coverage, the authors of [4] proposed an approach employing a SAT-based encoding for restricted symbolic logic (RSL) in order to model unknown values more precisely compared to three-valued logic. For an efficient encoding, structural knowledge of the circuit is exploited in order to consider only relevant X-reconvergencies. Similarly, [5] proposed (1) an SMT-like solving strategy for RSL with on-the-fly clause generation during solving and (2) an untestability check which allowed the accurate detection of untestable faults. An improved two-valued version of this accurate untestability check was used in [6] together with an enhanced SAT-based encoding that combines the accuracy of RSL with the efficiency of three-valued encodings.

Still, all these RSL-based algorithms suffer from the same problem that they are unable to determine the accurate result in presence of unknown values – as soon as no test pattern is found for a certain fault and the incorporated untestability check fails. Furthermore, all these earlier approaches use a standard SAT- or QBF-solver.

Contribution of this paper: In contrast, here, we focus on the accurate classification of each fault by employing a highly incremental approach tailored for this specific problem class. In detail we present:

1) a highly incremental CEGAR-based algorithm that is completely accurate in presence of unknown values.
2) several optimizations in order to reduce the runtime of the overall method significantly.
3) a modified and stripped down SAT-solver which is the fundament of the CEGAR-based algorithm.

Experimental results for large industrial circuits with up to 2 400 000 gates show the efficiency of the proposed method. Compared to a state-of-the-art commercial tool the fault-coverage is increased significantly.

The rest of the paper is organized as follows. Section II gives a short introduction into the terminology. Sections III and IV describe the proposed method and how it is embedded into the complete ATPG framework. Experimental results are discussed in Section V, before we conclude the paper in Section VI.

II. Preliminaries

A. X-Values

An X-value represents a well-defined but unknown binary value – either being logic-0 or logic-1. This excludes undefined values (e.g. caused by undefined voltage levels) and corresponds to the semantics of unknown values in [7]. X-sources are signals which generate unknown values – e.g. uncontrolled sequential elements or clock domain crossings. If such an X-source is reachable from a signal or gate, then this signal or gate is X-dependent.

B. Fault Detection Requirements

Similar to [8], we consider a stuck-at fault detectable if and only if there is a test pattern which makes the fault observable at at least one output independent of the X-values. A fault is observable if an output shows value v in the fault-free case while it shows $\neg v$ in the faulty case – for all possible assignments to the X-sources.

C. SAT

In this paper we concentrate on (extended) SAT-based techniques to generate test patterns – or prove the non-existence thereof (untestability). The Boolean satisfiability problem (SAT) asks the question if there exists for a given Boolean formula an assignment to its variables such that this formula evaluates to true. Programs handling this kind of problem are called SAT-solvers. We will only give a very brief summary of SAT, a comprehensive introduction can be found in [9].

Most modern SAT-solvers do not accept arbitrary Boolean formulas – instead they expect a formula to be in conjunctive normal form (CNF). A CNF is a conjunction of clauses. Each clause is a disjunction of literals and every literal is either a Boolean variable or its negation.

A natural representation of a combinatorial circuit is a directed acyclic graph (DAG). The Tseitin-transformation [10] allows to translate such a DAG into a CNF by introducing auxiliary variables for each signal.

Furthermore, many SAT-solvers allow *incremental solving*. This means the overall problem is partitioned into a conjunction of sub-problems. We start with a sub-problem and let the SAT-solver search for a solution. If already this sub-problem has no solution, then the overall problem has no solution as well – because it is a conjunction of the sub-problems. Otherwise, if the solver finds a solution, we add a further sub-problem. Regarding the CNF representation this means we just add further clauses to the CNF which was already passed to the SAT-solver. The SAT-solver is now called on the extended CNF and is allowed to re-use all the knowledge generated during the previous solving calls. This re-use speeds up the consecutive solve calls significantly. Furthermore, a CNF could be solved under *assumptions*. This means the SAT-solver assumes the given assumption-literals to be true and tries to find a satisfying assignment under this restriction.

D. SAT-based ATPG

Without X-values the test pattern generation for a given fault location could be mapped to a Boolean satisfiability problem as follows: a CNF is created which contains

1) the fault-free representation of the circuit,
2) the faulty representation of the circuit,
3) D-chains [11] to model the propagation paths, and
4) additional clauses which demand a difference at at least one output of the fault-free and faulty version of the circuit.

If this CNF is satisfiable, a test pattern is extracted. Otherwise, the fault is proven to be untestable.

In presence of X-values this direct mapping is no longer possible if an accurate evaluation of the X-sources is desired. Instead an iterative approach will be used which is described in the next section.

III. The Method

The method described in this section builds on the untestability check (see Section III-A) first presented in [5] and the accurate simulation algorithm proposed in [12] (see Section III-B). Furthermore, a modified SAT-solver is used that is tailored for this specific problem class (see Section III-E).

A. Accurate Untestability Check (AUC)

As stated in Section II-B, a fault is detectable if there is a test pattern which makes the fault observable at at least one output independent of the X-values. In other words: the test pattern has to give *the same* difference on an output *for all* X-values. This means if there *exists one* assignment to the X-sources such that no difference is propagatable to at least one output, then we have proven the untestability of the considered fault – because at least one assignment to the X-sources does not have a test pattern.

So the basic idea is to check if this necessary condition for the existence of a test pattern is violated. Obviously, this condition does not guarantee to detect all accurately untestable faults. But it can be easily strengthened. Instead of checking only one assignment to the X-sources, we check two or more simultaneously. This is done by

1) copying the X-dependent parts of the circuit,
2) applying different assignments to the X-sources in each copy, and
3) adding extra clauses to propagate the fault effect in all copies to the same output – while demanding that the fault-free case shows the same value v in all copies and the opposite value $\neg v$ in the faulty case.

B. Accurate Testability Check (ATC)

If the CNF created by the untestability check is unsatisfiable, the fault is proven to be untestable. But if the CNF is satisfiable, the valuation of the variables representing the controllable inputs could or could not contain a valid test pattern – we simply do not know without further investigation.

On the other hand, it is possible to check whether a pattern for the controllable inputs propagates a difference to a certain output o – so to say an accurate testability check. This check is also SAT-based. Similar to [12] we create a CNF which contains

1) the fault-free representation of the circuit,
2) the faulty representation of the circuit,
3) additional clauses which demand *no* difference at output o of the fault-free and faulty version of the circuit,
4) the test pattern to be checked as assignment to the variables representing the circuit inputs, and
5) the X-sources as Boolean variables.

When searching for a test pattern, we search for a valuation of the controllable inputs. In contrast, here the values of the controllable inputs are now given and we search for an assignment to the X-sources. If the created CNF is unsatisfiable, there are no assignments to the X-sources violating the detection condition. In other words: the test pattern is valid and propagates the fault-effect to output o – for all possible assignments to the X-sources. On the other hand, if this CNF is satisfiable, we get precious information regarding an assignment to the X-sources that violates the detection requirements. This information is used to build the accurate ATPG as described below.

C. The CEGAR-Loop

The key idea behind our accurate CEGAR-based algorithm lies in the clever combination of both checks described above. If the accurate untestability check (see Section III-A) succeeds (the AUC-CNF is unsatisfiable), the fault is proven to be untestable. If the AUC-CNF is satisfiable, we get a possible test pattern which may propagate a difference to one or more outputs. For each output with a difference, we can now use the accurate testability check (see Section III-B) in order to determine if the possible test pattern is a valid test pattern. If the ATC succeeds (the ATC-CNF is unsatisfiable) then the fault is indeed testable and we are done. If the ATC-CNF is satisfiable, we get a valuation of the X-sources which proves that the possible test pattern is a spurious test pattern[1]. We can use this assignment to the X-sources in order to exclude the spurious test pattern by refining AUC-CNF with a further copy. Then we solve again AUC-CNF. We iterate as long as we get spurious test patterns. Figure 1 depicts the principle. We operate on two CNF representations in an interleaving manner and do incremental SAT-solving. In each iteration we first solve AUC-CNF and – if needed – switch to ATC-CNF. Furthermore, AUC-CNF is extended with further clauses and the assignment to the X-sources determined by ATC-CNF.

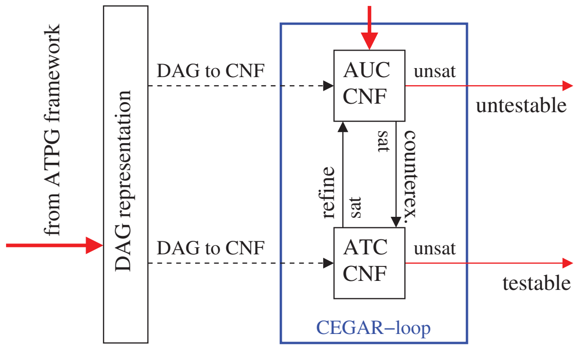

Fig. 1: In the CEGAR-loop, AUC-CNF and ATC-CNF are solved in an interleaving manner.

This iterative principle is also called *CounterExample Guided Abstraction Refinement* (CEGAR) [13]. We start with a coarse abstraction (AUC-CNF contains only one copy) and search for a counterexample (a possible test pattern). Then we check if the counterexample is spurious or not (possible test pattern is spurious or valid) and refine our abstraction in case of a spurious counterexample in order to exclude it (AUC-CNF is extended with further clauses). A similar approach was proposed in [14]. But while [14] was designed to solve generic 2-QBF formulas, we tailor our approach to X-aware ATPG and exploit the structural knowledge of the circuit as explained in Section III-D. Furthermore, we extend the above-described CEGAR-loop with a second method to exclude spurious counterexamples. We utilize the approach proposed in [15] in order to determine a small subset of the controllable inputs which provably represent a set of spurious test patterns. This set of spurious test patterns can be excluded by adding just a single clause to AUC-CNF. The calculation of such a small subset is in principle possible with a standard SAT-solver, but it can be made much more efficient with an adapted SAT-solver by exploiting the knowledge already present in the implication graph of the solver.

[1]This means, that for some valuations of the X-sources this pattern propagates a difference to the output – but there exists at least one valuation for which this pattern does not produce the desired output difference. Therefore, we call such a pattern a spurious test pattern.

D. *Efficient Handling of Multiple Outputs*

There could be thousands of outputs a fault might propagate a difference to. Therefore, we do not apply the CEGAR-loop directly to the full fault instance – instead we use a layered approach. The outer layer uses an extended version of the untestability check as a filter (FIL) in order to determine a possible test pattern and the outputs which might show an observable difference for this pattern. The inner layer consists of the CEGAR-loop which now considers only one output at a time. After solving FIL-CNF we already have a possible test pattern. Therefore, the CEGAR-loop is now entered at ATC-CNF as shown in Figure 2.

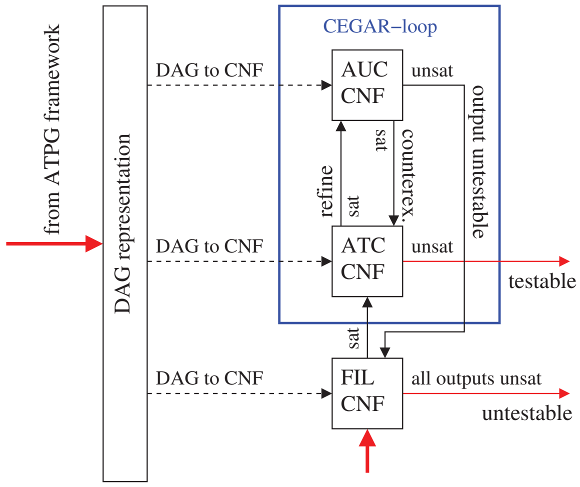

Fig. 2: Combination of FIL-CNF with the CEGAR-loop.

Again, we make use of incremental SAT-solving and create FIL-CNF step-by-step. In a nutshell: we first build FIL-CNF with all gates included in the cone of influence of the fault-site in order to check if the fault is sensitizable. If the fault is sensitizable, we build all gates in the cone of influence of the nearest output – the nearness of an output o is determined by the longest path from an input to o. If the fault is untestable at the current output we proceed with the second nearest output until either a test pattern is found or all outputs have been processed without finding a pattern.

A similar approach is state-of-the-art when doing SAT-based ATPG. Nonetheless, we need to make some adaptions which are not needed when doing two- or three-valued SAT-based ATPG – because we extend FIL-CNF not only in one, but in two dimensions. On the one hand we incrementally add new outputs – and on the other hand we also add further copies of X-dependent parts of the circuit. Additionally, we have to keep in mind that solving FIL-CNF is somewhat asymmetric: only unsatisfiability is a conclusive answer – whenever FIL-CNF is satisfiable the CEGAR-loop has to be called.

Therefore, when checking if a fault is sensitizable, we are done if FIL-CNF is unsatisfiable. In case it is satisfiable we employ the CEGAR-loop to get a conclusive answer. The same is done for each output: if FIL-CNF is unsatisfiable for an output, we proceed with the next output; if it is satisfiable, we employ the CEGAR-loop – which could either return a test pattern or prove that the fault is untestable at the current output. In standard SAT-based ATPG algorithms, when a fault could not be propagated to a certain output, a unit clause is added to the CNF which encodes the knowledge that no difference will ever be observed at this output. This is done to prune the search space of the SAT-solver before proceeding with the next output. Regarding FIL-CNF this is only allowed if FIL-CNF is unsatisfiable by itself and a call to the CEGAR-loop was not needed – otherwise no such information can be added to FIL-CNF. One has to keep in mind that the CEGAR-loop is only called if FIL-CNF is satisfiable. If we now encode the information into FIL-CNF that no difference will ever be observed at the current output, we would implicitly forbid all those test patterns which are spurious for the current output. But there is no guarantee that all these patterns are also spurious for the outputs to be processed later on. In order to reduce such cases, we add further copies of X-dependent parts of the circuit to FIL-CNF whenever the CEGAR-loop proved the untestability of the fault on the current output. FIL-CNF starts with two copies and may have up to four copies with different and heuristically chosen assignments for the X-sources for each copy.

The aim of this layered approach is to quickly isolate outputs which are candidates according to our detection requirement. To give an impression about how well this filtering works, here some numbers from our experiments for a specific circuit: assume after handling all easy-to-detect faults, 45 000 faults remain unclassified. Further assume each of these faults could affect 250 outputs on average – resulting in 11 250 000 outputs to be visited in worst case[2]. If we now apply our approach, it visits 4 500 000 outputs in total, but calls the CEGAR-loop only 32 000 times. The CEGAR-loop returns a test pattern in 16 000 cases and proves the remaining 16 000 cases to be untestable. This means over 99% of the visited outputs are already filtered out by FIL-CNF and are not passed to the CEGAR-loop.

As mentioned above, FIL-CNF is build incrementally for a fault. In contrast, AUC-CNF and ATC-CNF are build from scratch whenever the CEGAR-loop is called, because the CEGAR-loop only considers one output and all gates in the cone of influence of this output. Nonetheless, the CEGAR-loop is incremental itself, because in every iteration AUC-CNF is extended with a further copy of X-dependent circuit parts. Furthermore, when the CEGAR-loop is entered, ATC-CNF is created once and is re-used in every iteration, because the patterns are applied as assumptions.

E. SAT-solver Modifications

In this paper we use an optimized and stripped down version of the solver core of iSAT3 [16, 17]. We did several modifications in order to tailor the solver for our method:

1) In order to allow the efficient solving of several hundred fault instances per second, we optimized the way clauses are stored. Now it is a very cheap operation to discard a complete CNF and to reset the solver.
2) Clauses satisfied on decision level 0 are removed from the watch-lists immediately instead of using a periodic cleanup function.
3) As already mentioned in Section III-C, we did adaptions in order to efficiently determine a small subset of the controllable inputs which provably represent a set of spurious test patterns. Here, we exploit the knowledge already present in the implication graph of the solver. We omit further details here due to lack of space.

Additionally, instead of a timeout, we limit the number of conflicts, decisions and deductions the SAT-solver is allowed to perform. We apply a global limit per fault instance and a local limit per solve call. If one of these limits is reached the solving process is aborted and the fault stays unclassified. Using these limits instead of a timeout allows a deterministic behaviour regarding the resulting fault coverage – independent of the actual speed of the CPU the solver is running on.

IV. The complete ATPG Framework

The complete ATPG framework uses the accurate fault simulation described in [8] and the simulator optimizations presented in [6]. The ATPG framework reads the circuit description in Verilog and generates a fault list. Instead of random patterns, we use a commercial tool to filter out easy-to-detect faults (see Section V). The remaining faults are then processed one-by-one. All relevant gates needed for sensitisation, justification and propagation of a fault, along with the fault location are passed as a DAG to our approach (see Section III-D). Our method classifies a fault either as untestable, testable or aborted. A fault is aborted if the given limits for the search process are reached (see Section III-E). If a fault is testable, we employ the accurate simulator for fault dropping.

V. Results

The proposed algorithms are implemented in C. We evaluate our method for stuck-at faults on full-scan circuits of the largest ISCAS benchmarks as well as larger industrial designs from NXP. All experiments were conducted on an Intel Xeon CPU with 3.3 GHz. We perform structural fault collapsing prior to each ATPG run and assume that a fixed and randomly selected subset of circuit inputs is used as X-sources[3].

Experiment 1: In this experiment, we consider for each circuit an X-ratio of 1%, 2% and 5%. For every X-ratio we consider five different X-source assignments. The reported results in Table I are the rounded average over these five assignments per circuit. In order to remove all easy-to-detect faults prior to each ATPG run, all faults detectable by the used commercial tool are marked as detected and are not subject to further processing. Therefore, the runtime of the commercial tool is always included in the runtime of the proposed CEGAR-based ATPG in Table I. Furthermore, we listed the runtime of the commercial tool separately in order to show its fraction of the total runtime.

The proposed CEGAR-based ATPG is able to increase the fault coverage (FC) considerably compared to the commercial tool e.g. for an X-ratio of 5% the biggest gains are (given in percent points): c6288 +34.25%pt., p78k +8.01%pt. and p378k +7.98%pt.. On average over all considered NXP designs and an X-ratio of 5% the FC of our method achieves a 3.79%pt. higher FC than the commercial tool.

[2]Because testable faults usually do not require to visit all outputs which might be affected by the fault, the actual number of visited outputs will be lower.

[3]The effect of clustered X-sources in ATPG has been discussed in [3].

Because our method is accurate and does not introduce any pessimism, the percentage of aborted faults is equal to the number of unclassified faults. For many circuits and X-ratios, our method classifies over 99.99% of all faults. On average less than 0.04% of the faults were aborted.

Experiment 2: While the first approach for accurate X-aware ATPG presented in [3] uses an off-the-shelf QDPLL-based QBF-solver, the method presented here is specifically tailored for this problem class. Therefore, we do no explicit comparison with this previous approach – instead we compare against the QBF-solver RAReQS [18] which is a generalized version of [14]. RAReQS is CEGAR-based as well and therefore more closely related to our approach than a QDPLL-based solver. Furthermore, RAReQS outperformed all other QBF-solvers for application instances in the last QBF competition 2014.

We selected four circuits for this experiment: p78k, p141k, p267k and p388k. Additionally, we used an X-ratio of 5% with one X-source assignment for each circuit. In order to compare our method to RAReQS, we wrote out every instance which was passed to our approach to a file and let RAReQS solve this file with a timeout of 2 seconds. The runtime of our approach also includes the time which was needed to write out all the instances. The runtime of RAReQS contains the cumulated execution time for all fault instances[4].

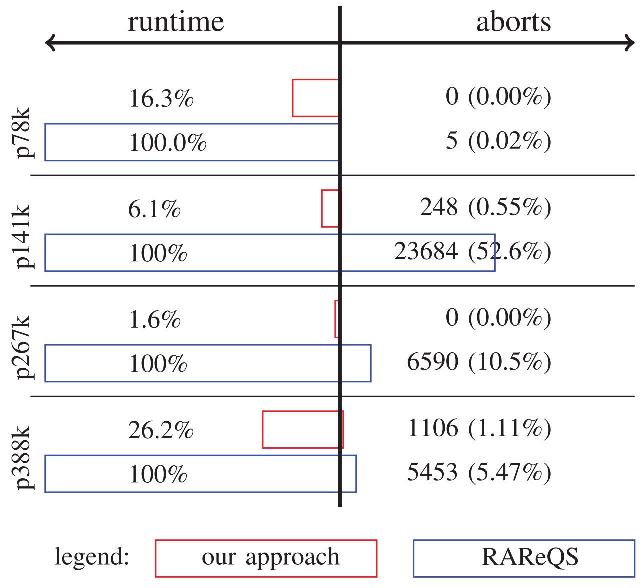

Fig. 3: Comparison of the runtime and the aborted faults between our approach and RAReQS for different industrial designs with an X-ratio of 5% (smaller bars are better).

Figure 3 shows a bar chart which compares on the left side the runtimes in a relative manner – e.g. for p141k our approach only needs 6.1% of the runtime of RAReQS. On the right side of Figure 3 we compare the number of aborted fault instances in absolute and relative numbers – e.g. for p78k our approach is able to classify all faults accurately while RAReQS has a few aborts. It can be observed that our method always outperforms RAReQS in terms of runtime and number of aborted faults, e.g. for p267k our approach is more than 50× faster and classifies every fault, while RAReQS aborts 10.5% of the faults.

[4]Of course each execution of a program binary causes additional runtime overhead – on the other hand the observed runtime differences between our approach and RAReQS are way too big in order to attribute them solely to this overhead.

While incremental SAT-solving is a well established technique and is used in numerous applications, incremental QBF-solving is in general subject to several limitations, because adding new clauses to the CNF and variables to the quantifier-prefix may invalidate already learnt knowledge – resulting in a loss of efficiency. Our approach does not have such limitations, because it heavily relies on incremenal SAT-solving. This is particularly exploited during the construction of FIL-CNF and also reflected in Figure 3. Compared to RAReQS the efficiency of our layered approach increases with the number of outputs a fault might be propageted to. An extreme example is p141k: our approach classifies more than 99% of the faults, while RAReQS classifies less than 50%. Furthermore, our approach needs less then a tenth of the runtime which is required by RAReQS. The following table gives an overview of the percentage of the faults that could affect 1, 2-10, 11-100, 101-1000 or more than 1000 outputs. As shown, in case of p141k, more than 20% of the faults could affect 100 or more outputs.

circuit	number of outputs a fault might be propagated to				
	1	2-10	11-100	101-1000	1001+
p78k	17.82%	31.84%	50.34%	-	-
p141k	41.24%	27.36%	11.31%	14.09%	6.01%
p267k	37.05%	33.46%	19.62%	9.49%	0.38%
p388k	38.51%	21.86%	35.61%	3.42%	0.60%

This means the more outputs a fault might affect, the stronger will be the effect of FIL-CNF. This underlines the effectiveness of tailoring our approach for X-aware ATPG – because standard solvers with non-incremental solving are unable to exploit this problem-specific knowledge.

VI. Conclusion

In this paper we presented a highly incremental SAT-based CEGAR approach tailored for accurate ATPG in presence of unknown values: It combines the benefits of using a SAT-solver adapted to the specific problem with a layer-based handling of the CEGAR-loop.

The results confirm the effectiveness of the presented method. Compared to a standard CEGAR-based QBF-solver, the proposed method is up to 50× faster. In comparison to a state-of-the-art commercial tool and an X-ratio of 5%, the fault coverage could be increased significantly by 4.72% on average. Furthermore, on average 99.96% of all faults of a circuit were classified accurately. At the same time, the runtime stays reasonable – even though circuits with up to 2 400 000 gates have been considered.

In the future we want to apply the presented approach to closely related problem classes such as finding initialization sequences or reducing the pattern count.

Acknowledgments

The authors thank Linus Feiten from the University of Freiburg and Michael Kochte from the University of Stuttgart for supporting this work. This work was supported by the German Research Foundation (DFG) under grant SFB/TR14 AVACS.

TABLE I: Results of the proposed CEGAR-based ATPG compared to a state-of-the-art commercial tool.

Circuit	Gates	Faults	X-Ratio	**Commercial tool**		**proposed CEGAR-based ATPG**				
			[%]	FC [%]	Runtime [s]	**ΔFC [%pt.]**	FC [%]	UT [%]	Aborts [%]	Runtime [s]
c6288	2 416	8 704	1.0	82.46	42	**14.92**	97.38	2.59	0.02	45
			2.0	— only one X-source as with X-Ratio 1.0% – therefore, the same results as above —						
			5.0	60.90	81	**34.25**	95.04	4.92	0.03	88
c7552	4 043	10 816	1.0	91.78	4	**0.99**	92.77	7.23	0.00	5
			2.0	88.50	5	**1.73**	90.23	9.77	0.00	6
			5.0	68.01	9	**6.48**	74.50	25.50	0.00	13
cs38417	23 537	59 041	1.0	95.54	1	**0.17**	95.71	4.29	0.00	2
			2.0	93.58	1	**0.25**	93.83	6.17	0.00	3
			5.0	86.54	1	**0.45**	86.99	13.01	0.00	6
p78k	74 243	225 476	1.0	97.29	10	**1.40**	98.69	1.31	0.00	14
			2.0	93.59	20	**3.44**	97.04	2.96	0.00	37
			5.0	84.24	38	**8.01**	92.25	7.75	0.00	222
p89k	88 726	239 090	1.0	91.94	78	**0.17**	92.11	7.89	0.00	133
			2.0	86.32	84	**0.27**	86.59	13.41	0.00	169
			5.0	71.74	38	**1.04**	72.77	27.23	0.00	180
p100k	96 685	259 322	1.0	95.28	121	**0.89**	96.17	3.83	0.00	212
			2.0	91.40	181	**2.15**	93.55	6.44	0.01	339
			5.0	80.29	468	**3.81**	84.10	15.66	0.24	1 927
p141k	172 686	452 599	1.0	95.86	40	**0.38**	96.24	3.76	0.00	641
			2.0	93.90	44	**0.62**	94.52	5.48	0.00	1 051
			5.0	85.82	77	**1.89**	87.71	12.24	0.05	2 783
p267k	271 538	658 395	1.0	94.71	42	**0.35**	95.06	4.94	0.00	160
			2.0	90.95	49	**0.43**	91.38	8.62	0.00	285
			5.0	80.00	84	**2.03**	82.02	17.94	0.03	616
p378k	371 215	1 127 364	1.0	96.83	65	**1.70**	98.53	1.47	0.00	162
			2.0	93.59	104	**3.42**	97.00	3.00	0.00	315
			5.0	84.46	195	**7.98**	92.44	7.56	0.00	1 483
p388k	489 271	1 352 303	1.0	93.86	360	**3.53**	97.39	2.61	0.00	1 419
			2.0	91.71	634	**3.93**	95.64	4.34	0.02	2 323
			5.0	83.03	2 729	**5.39**	88.42	11.41	0.17	10 109
p951k	1 002 883	2 539 361	1.0	96.39	474	**0.36**	96.74	3.25	0.01	2 413
			2.0	93.69	797	**0.63**	94.32	5.66	0.02	4 157
			5.0	85.28	1 107	**1.46**	86.75	13.21	0.05	9 587
p1522k	1 104 085	2 835 817	1.0	89.77	1 596	**0.95**	90.72	9.24	0.04	11 741
			2.0	84.08	2 251	**1.46**	85.54	14.38	0.09	18 940
			5.0	70.63	3 606	**3.61**	74.25	24.45	0.30	46 556
p2927k	2 408 312	5 697 626	1.0	93.20	6 162	**0.55**	93.75	6.23	0.02	16 901
			2.0	88.60	11 547	**1.13**	89.74	10.20	0.07	30 722
			5.0	77.06	21 621	**2.67**	79.74	19.22	0.27	73 570

References

[1] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM J. Res. Dev.*, vol. 10, no. 4, pp. 278–291, july 1966.

[2] A. Jain, V. Boppana *et al.*, "Testing, verification, and diagnosis in the presence of unknowns," in *VTS*, 2000.

[3] D. Erb, M. Kochte *et al.*, "Accurate qbf-based test pattern generation in presence of unknown values," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2015.

[4] D. Erb, K. Scheibler *et al.*, "Test pattern generation in presence of unknown values based on restricted symbolic logic," in *Proc. of the IEEE International Test Conference (ITC'14)*, 2014.

[5] K. Scheibler, D. Erb, and B. Becker, "Improving test pattern generation in presence of unknown values beyond restricted symbolic logic," in *20th IEEE European Test Symposium, ETS 2015*. IEEE, 2015.

[6] D. Erb, K. Scheibler *et al.*, "Mixed 01X-RSL-Encoding for Fast and Accurate ATPG with Unknowns," in *to appear in Proc. of IEEE Asia and South Pacific Design Automation Conference*, 2016.

[7] S. C. Kleene, *Introduction to Metamathematics*. North-Holland Publishing Co., Amsterdam, 1952.

[8] D. Erb, M. A. Kochte *et al.*, "Exact logic and fault simulation in presence of unknowns," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 19, no. 3, Jun. 2014.

[9] H. K. Büning and U. Bubeck, *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications 185. IOS Press, 2009.

[10] G. Tseitin, "On the complexity of derivation in propositional calculus," *Studies in constructive mathematics and mathematical logic*, 1968.

[11] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. CAD*, vol. 11, no. 1, pp. 4–15, jan 1992.

[12] M. A. Kochte, M. Elm, and H.-J. Wunderlich, "Accurate X-propagation for test applications by SAT-based reasoning," *IEEE Trans. CAD*, vol. 31, no. 12, pp. 1908–1919, 2012.

[13] E. M. Clarke, O. Grumberg *et al.*, "Counterexample-guided abstraction refinement for symbolic model checking," *J. ACM*, vol. 50, no. 5, pp. 752–794, 2003. [Online]. Available: http://doi.acm.org/10.1145/876638.876643

[14] M. Janota and J. P. M. Silva, "Abstraction-based algorithm for 2QBF," in *Theory and Applications of Satisfiability Testing - SAT 2011*, ser. Lecture Notes in Computer Science, K. A. Sakallah and L. Simon, Eds., 2011.

[15] K. Ravi and F. Somenzi, "Minimal assignments for bounded model checking," in *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004*, K. Jensen and A. Podelski, Eds., 2004, pp. 31–45.

[16] K. Scheibler, S. Kupferschmid, and B. Becker, "Recent improvements in the SMT solver iSAT," in *MBMV'13*, 2013, pp. 231–241.

[17] K. Scheibler and B. Becker, "Using interval constraint propagation for pseudo-boolean constraint solving," in *FMCAD*, 2014.

[18] M. Janota, W. Klieber *et al.*, "Solving QBF with counterexample guided refinement," in *SAT*, 2012.