Test Pattern Generation in Presence of Unknown Values Based on Restricted Symbolic Logic

Erb, Dominik; Scheibler, Karsten; Kochte, Michael A.; Sauer, Matthias; Wunderlich, Hans-Joachim; Becker, Bernd

Proceedings of the IEEE International Test Conference (ITC'14) Seattle, Washington, USA, 20-23 October 2014

doi: http://dx.doi.org/10.1109/TEST.2014.7035350

Abstract: Test generation algorithms based on standard n-valued logic algebras are pessimistic in presence of unknown (X) values, overestimate the number of signals with X-values and underestimate fault coverage. Recently, an ATPG algorithm based on quantified Boolean formula (QBF) has been presented, which is accurate in presence of X-values but has limits with respect to runtime, scalability and robustness. In this paper, we consider ATPG based on restricted symbolic logic (RSL) and demonstrate its potential. We introduce a complete RSL ATPG exploiting the full potential of RSL in ATPG. Experimental results demonstrate that RSL ATPG significantly increases fault coverage over classical algorithms and provides results very close to the accurate QBF-based algorithm. An optimized version of RSL ATPG (together with accurate fault simulation) is up to 618x faster than the QBF-based solution, more scalable and more robust.

Preprint

Test Pattern Generation in Presence of Unknown Values Based on Restricted Symbolic Logic

Dominik Erb*, Karsten Scheibler*, Michael A. Kochte‡, Matthias Sauer*, Hans-Joachim Wunderlich‡, Bernd Becker*
*University of Freiburg, Georges-Köhler-Allee 51, 79110 Freiburg, Germany
‡University of Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany

***Abstract*—Test generation algorithms based on standard n-valued logic algebras are pessimistic in presence of unknown (X) values, overestimate the number of signals with X-values and underestimate fault coverage.**

Recently, an ATPG algorithm based on quantified Boolean formula (QBF) has been presented, which is *accurate* in presence of X-values but has limits with respect to runtime, scalability and robustness.

In this paper, we consider ATPG based on restricted symbolic logic (RSL) and demonstrate its potential. We introduce a *complete* RSL ATPG exploiting the full potential of RSL in ATPG. Experimental results demonstrate that RSL ATPG significantly increases fault coverage over classical algorithms and provides results very close to the accurate QBF-based algorithm. An optimized version of RSL ATPG (together with accurate fault simulation) is up to 618× faster than the QBF-based solution, more scalable and more robust.

***Index Terms*—SAT, QBF, test generation, ATPG, Unknown values, Restricted symbolic logic**

I. Introduction

During test application, unknown values (X-values) may occur at uncontrolled memory elements (uninitialized memory arrays, non-scan flipflops or latches), clock-domain boundaries or at analog-digital converters.

X-values reduce observability and controllability of signals during testing, which results in a reduced fault coverage. However, due to the performance and area overhead of design-for-test structures, not all X-values can be avoided by X-blocking hardware.

Test generation and fault simulation is typically performed based on logic algebras with a low fixed number of symbols to distinguish between fault-free, faulty and unknown signal values. In the five-valued D-calculus [1], a single X-symbol is used to represent an unknown value in the fault-free or faulty circuit. The accuracy has been increased by the nine-valued algebra [2], which uses three values $\{0, 1, \mathrm{X}\}$ both for the fault-free and faulty circuit. This algebra can also be used in SAT-based test generation algorithms [3]. The 16-valued algebra [4] for test generation further distinguishes between unknown signal values with equivalent or complementary states in the fault-free and faulty circuit.

However, these algebras do not overcome the fundamental problem that *different* X-values are not clearly identified and distinguished. Thus, such multi-valued modeling still causes pessimistic results when multiple X-sources exist in the circuit, and X-valued signals reconverge.

The symbolic representation of signal values, e. g. by binary decision diagrams [5], allows to accurately express the dependency on the X-sources. This, however, may incur prohibitively high memory requirements.

In *restricted symbolic logic* (RSL), different symbols are used to represent X-values and their negation. In logic simulation based on RSL [6] (also known as numbered-X or indexed simulation) this allows to accurately evaluate simple reconvergences of X-valued signals. In [7], the idea of using multiple X-symbols is discussed for the first time. The use of RSL during the *forward implication* step in topological test pattern generation is proposed in [8, 9]. Yet, there has been no complete ATPG algorithm that also uses RSL during the *backward implication* step.

Figure 1 shows a circuit with two X-sources (signals b and d). In this circuit, the stuck-at-0 fault at output j is untestable when using three-valued logic since j cannot be justified to logic-1. Using restricted symbolic logic, the fault is detected by the pattern $(a, c, e) = (1, 0, 1)$, since the X-values stemming from signal b reconverge at gate G_5, cancel each other out (X-canceling) and generate a logic-1 value at signal j.

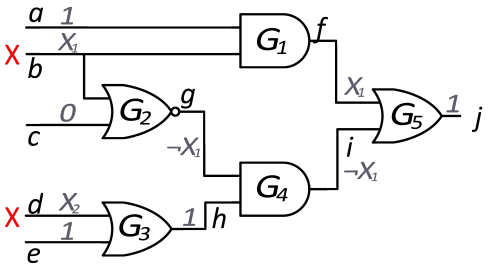


Fig. 1: Stuck-at-0 fault at output j is undetectable in three-valued modeling, but detectable using restricted symbolic logic.

Still, restricted symbolic logic is pessimistic when X-values from *different* X-sources converge at a signal.

Accurate logic and fault simulation utilizing SAT-based reasoning have recently been proposed in [10, 11]. An accurate ATPG algorithm based on quantified Boolean formulas (QBF), which is able to generate a test pattern for each testable stuck-at fault or prove its untestability, is proposed in [12]. However, deciding about the satisfiability of a QBF is a PSPACE-

complete problem, which is more complex than classical NP-complete topological or SAT-based ATPG [13]. Thus, the increased accuracy in ATPG causes considerably higher runtime and low scalability.

In this paper we present:

- to the best of our knowledge the first ATPG with the accuracy of restricted symbolic logic, called *complete RSL ATPG*, that uses restricted symbolic logic both during forward and backward implications.
- an extended version of D-chains [14] to model propagation paths of fault effects in case RSL is used (Section III-D).
- a further *optimized RSL ATPG* algorithm that significantly reduces runtime at the cost of only slightly increased pessimism compared to the complete RSL approach (Section III-D,III-F). This pessimism is compensated by incorporating accurate fault simulation [10].
- a thorough comparison of the proposed algorithms with classical three-valued ATPG and accurate QBF-based ATPG [12, 15] in presence of X-values.

The experimental results demonstrate that the RSL-based ATPG algorithms increase fault coverage significantly compared to three-valued algorithms and provide a result very close to an accurate QBF-based ATPG at significantly lower runtime: The *complete RSL ATPG* is one order of magnitude faster than the QBF-based algorithm. The combination of *optimized RSL ATPG and accurate fault simulation* further increases the quality. It is up to two orders of magnitude faster and has much less aborted faults.

The paper is structured as follows: the next section introduces the used terminology and fault detection requirements. Section III presents the proposed RSL ATPG algorithms, followed by the full ATPG framework in Section IV. Experimental results are discussed in Section V.

II. Terminology and Problem Statement

A. X-Values

A signal with an X-value carries a binary value of logic-0 or logic-1, but it is not known which one. This excludes undefined voltage levels caused for instance by driver contention. This corresponds to the semantics of unknown values in Kleene's strong three-valued logic [16]. The signals which generate X-values in the circuit are called X-sources; e. g. uncontrollable non-scan flip-flops or other uninitialized memory elements. A signal or gate in the circuit is X-dependent if and only if it is reachable from an X-source.

B. Two-, Three-Valued and Restricted Symbolic Logic (RSL)

The two-valued (switching) logic distinguishes the two binary values logic-0 and logic-1. It is extended by a third value X to denote unknown values in the three-valued logic.

The restricted symbolic logic [6] (RSL) extends this value domain by *named and distinguishable* X-values and their negation. Restricted symbolic logic accurately evaluates simple, local X-reconvergences and the resulting canceling of X-values. If X-values from different X-sources converge at a gate, and the output value is not determined by a controlling value at an input or by X-canceling, the output is assigned a new, unique X-symbol. This new X-symbol has no correlation with the input values any more and therefore introduces pessimism into the evaluation.

In fault simulation and ATPG, two-, three-valued and restricted symbolic logic can be used to model the signal values in the fault-free and faulty circuit.

C. Value Encoding for SAT-Based ATPG

In this work, we map the test pattern generation to a Boolean satisfiability (SAT) problem. Most modern SAT-solvers expect a conjunctive normal form (CNF) as input. A CNF is a conjunction of clauses with each clause being a disjunction of literals – each literal is either a Boolean variable or its negation.

The size of the CNF encoding highly depends on the number of distinguishable values a signal should be able to carry. In two-valued logic, the possible values are logic-0 and logic-1. Therefore, one literal is sufficient for each signal. In order to handle unknowns, three-valued logic additionally supports an X-value. This requires two literals l_1, l_2 to encode the possible signal values. Usually, logic-0 and logic-1 are encoded as $l_1 = 0, l_2 = 1$ and $l_1 = 1, l_2 = 0$, whereas an unknown value is encoded as $l_1 = 0, l_2 = 0$ [3]. The combination $l_1 = 1, l_2 = 1$ is forbidden by adding the clause $(\neg l_1 \vee \neg l_2)$ for any signal. Compared to two-valued logic, the number of clauses for three-valued encoding is more or less doubled.

Regarding restricted symbolic logic, each signal may carry a different number of distinguishable X-values: X_1, ..., X_m. Furthermore, we distinguish between an X-value X_i and its negation $\neg\mathrm{X}_i$.

The signal encoding with literals directly influences the number of clauses needed to represent a gate. It is important that the encoding allows an efficient comparison of X-values: when a gate has multiple X-values at its inputs, we have to determine whether there are X-values with the same index. Therefore, we encode the index of the X-value in binary representation. This requires $\lceil \log_2 m \rceil$ literals (in the following named b-literals) in order to be able to represent m different X-values. There are two further literals: the n-literal encodes if a signal is negated or not, and the x-literal informs whether the signal carries an X-value. The following table summarizes the different literal types in our encoding:

Literal	Meaning
n	$n = 1$: signal is negated
x	$x = 1$: signal has X-value
b_k	X_i $i = \{1, \ldots, m\}$ binary encoded, $k \in \{1, \ldots, \lceil \log_2 m \rceil\}$

Furthermore, this encoding easily allows the handling of two-valued signals, because a two-valued signal is just a special case: the n-literal represents the logic value of the signal, the x-literal is always 0 and with $x = 0$ the values of the b-literals are irrelevant. Hence, logic-0 is represented as $n = 0, x = 0$

and logic-1 as $n = 1, x = 0$. In fact, our application simplifies the encoding by omitting the x- and b-literals in such cases.

When referring to a literal l of signal s, we use the notation $s[l]$, e. g. $i1[n]$ for the n-literal of signal $i1$.[1]

D. Fault Detection Requirements

A stuck-at fault f is detected if and only if an output $o \in O$ exists at which the fault is observable independent of the state of the X-sources. Let $v^G(p, s)$ and $v^f(p, s)$ return the value of a signal s under a pattern p in the fault-free circuit and the circuit under fault f. Then, fault f is detected if and only if $\exists o \in O: \; v^G(p, o), v^f(p, o) \in \{1, 0\} \wedge v^G(p, o) \neq v^f(p, o)$. A fault for which no test pattern can be generated using restricted symbolic logic is called *RSL-untestable*.

III. ATPG Based on Restricted Symbolic Logic

The next section gives a brief overview of the algorithm, followed by a detailed discussion.

A. Overview

For each considered fault, a SAT instance is constructed and evaluated by a SAT-solver. Prior to constructing the SAT instances, each X-dependent gate is preprocessed to ensure an efficient encoding (cf. Section III-B). Afterwards, all gates required to adjust the fault-free value at the faulty signal (the so called adjustment cone) are encoded as explained in Section III-C.

If the instance cannot be satisfied, i. e. the fault-free value cannot be justified at the fault site, the fault is not considered any further. Otherwise, each output reachable from the fault site is considered one by one, starting with the output having the fewest X-dependent signals and gates – i. e. the adjustment cone of each output is modeled. For all gates which may be affected by the considered fault, a fault-free and faulty version of each gate is encoded. For all other required gates, the fault-free version is sufficient.

The SAT instance is augmented by difference literals (D-literals) and D-chains to speed up the search for a test pattern as explained in Section III-D. Finally, a constraint enforcing a difference between at least one output in the fault-free and faulty version is added to the instance. If the SAT-solver finds a satisfying assignment, the fault is detected. The test pattern is extracted, and fault simulation is performed with the pattern to find all faults detectable by this pattern and exclude them from further pattern generation. Otherwise, a test pattern derived using restricted symbolic logic does not exist.

B. Gate Preprocessing for Efficient Encoding

In principle, each gate depending on an X-source may lead to a new X-value which needs to be considered at the output of the corresponding gate. This would require to encode each of these signals using the previously described two literals for the n-, x-literals, as well as a rather large number of b-literals to be able to represent each possible X-value – leading to a tremendously increased number of clauses. Yet, the structure of the circuit allows to limit the number of X-values to be considered at a signal which enables a much more compact representation of the modeled signals and gates.

In this work, an efficient restricted symbolic representation of each fault instance is achieved through preprocessing of each gate. This avoids unnecessary encoding overhead since only those cases are modeled that can occur in the circuit. For each gate, the X-dependency of each input is computed. For gates with one or multiple X-dependent inputs, the circuit structure is analyzed to find gate output values that cannot occur and thus, are excluded from the gate encoding:
(1) At gates with only one X-dependent input, the output only shows a logic value if (i) an X-independent input has the controlling value of the gate, or (ii) the X-dependent input shows no X-value. Otherwise, the output carries the same X-value as the X-dependent input.
(2) For gates with more than one X-dependent input, the possible output values depend on the input values as follows:

- If an X-value X_i and its negation $\neg X_i$ are present at more than one input, the output of an AND-gate (NAND-gate) will show a logic-0 (1). In that case, an OR- or two-input XOR-gate (NOR- or two-input XNOR-gate) will show a logic-1 (0).
- If the same X-value is present at all inputs of a two-input XOR-gate (XNOR-gate), it will generate a logic-0 (1).
- If none of the inputs with logic values has the controlling value, and all X-dependent inputs carry the same X-value X_i, then the output has the identical X-value X_i.
- If different X-values are present at the inputs and no input has the controlling value of the gate, a new X-value emerges at the output according to restricted symbolic logic.

Figure 2 shows an example with two X-sources X_1, X_2, and five X-dependent gates $G_1, \ldots, G_5$. Obviously, the X-sources b,d can only carry an X-value, while all other inputs a,c,e may show a logic-0 or logic-1.

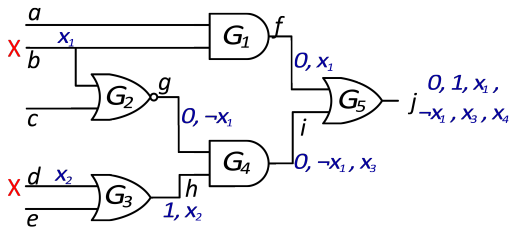


Fig. 2: Result of the preprocessing.

In the example, gate G_1 only depends on one X-source X_1. Therefore, only logic-0 (if $a = 0$) or the X-value X_1 (if $a = 1$) can appear at the output of G_1. This is also the case for gate G_2 where only logic-0 (if $c = 1$) or the X-value $\neg X_1$ (if $c = 0$) may be present at the output. At the output of gate G_3, logic-1 (if $e = 1$) or the X-value X_2 (if $e = 0$) can appear.

At gate G_4, both inputs are X-dependent. Its output may show the logic value 0 (if $g = 0$), the X-value $\neg X_1$ (if $g = \neg X_1$ and $h = 1$), or a new X-value X_3 (if both inputs

[1] An example for the proposed encoding is given in the Appendix.

show an X-value). Therefore, by taking the circuit structure into account, neither X_1, X_2, nor $\neg X_2$ are possible at the output of this gate. The same reasoning applies to the gate directly located at the output (G_5). At its output j, the X-values $X_2, \neg X_2$ and $\neg X_3$ are impossible and are excluded from the encoding.

Furthermore, the preprocessing may also lead to several already set/determined b-literals. For gate G_4 in the example of Figure 2, the output signal i can only show a logic-0 or the X-values $\neg X_1$, X_3. Thus, the number of b-literals required to encode the possible X-values of this signal is two. According to Section II, and using two b-literals b_2, b_1 for signal i, $i = X_1$ is expressed as $b_2 = 0, b_1 = 1$ and $i = X_3$ as $b_2 = 1, b_1 = 1$. Therefore, b_1 is always set to 1 as no valid assignment exists for which b_1 is set to 0.

According to this preprocessing, neither the stuck-at-0 fault at signal g nor the stuck-at-1 fault at signal h are testable based on restricted symbolic logic since for both faults the required value in the fault-free circuit cannot be adjusted. Consequently, these faults are already known to be RSL-untestable after preprocessing and do not need to be considered further. Also, a fault is only detectable if for a given pattern at least one output shows a different logic value comparing the fault-free and faulty circuit. Thus, also all outputs that cannot show a logic difference can be ignored during test pattern generation. If none of the outputs reachable from a faulty signal can show a logic difference, the fault is also classified as RSL-untestable.

C. Encoding Gates using Restricted Symbolic Logic

During CNF encoding we exploit the knowledge gathered during preprocessing by distinguishing between the following three cases. Note that in this paper, for simplicity of presentation, we only consider gates with two inputs. However, the encoding can be extended to gates with more inputs.

1) No input of a gate will see an X-value. In this case the encoding of the gate is similar to the encoding for two-valued logic – only the values of the n-literals are of interest here. For example, a two-input AND-gate with inputs $i1, i2$ and output o has the following implications:

$$\begin{array}{rcl} \neg i1[n] & \rightarrow & \neg o[n] \\ \neg i2[n] & \rightarrow & \neg o[n] \\ i1[n] \wedge i2[n] & \rightarrow & o[n] \end{array}$$

resulting in the following CNF representation:

$$(i1[n] \vee \neg o[n]) \ \wedge \ (i2[n] \vee \neg o[n]) \ \wedge \\ (\neg i1[n] \vee \neg i2[n] \vee o[n])$$

2) One input of a gate may see X-values. Regardless of the function that is realized by the actual gate, only these values are possible at the output: $0, 1, X_i$ or $\neg X_i$ (with X_i being the actual X-value seen on one input). In particular this means that the b-literals stay unchanged and will be shared between the input which may show an X-value and the output of the gate. Assume a two-input AND-gate may have an X-value on $i1$ and is two-valued on $i2$. This results in the following implications:

$$\begin{array}{rcl} i2[n] & \rightarrow & (i1[n] \leftrightarrow o[n]) \wedge (i1[x] \leftrightarrow o[x]) \\ \neg i2[n] & \rightarrow & \neg o[n] \wedge \neg o[x] \end{array}$$

yielding these six clauses (after applying some CNF level simplifications):

$$(\neg i2[n] \vee o[n] \vee \neg i1[n]) \ \wedge \ (\neg o[n] \vee i1[n]) \wedge \\ (\neg i2[n] \vee o[x] \vee \neg i1[x]) \ \wedge \ (\neg o[x] \vee i1[x]) \wedge \\ (i2[n] \vee \neg o[n]) \ \wedge \ (i2[n] \vee \neg o[x])$$

3) Two or more inputs of a gate may see X-values. Here, the basic idea is to do a case distinction depending on the actual values seen on the inputs. If both inputs show an X-value, it is checked if the b-literals of both inputs are equal, i.e. $\forall k(i1[b_k] \leftrightarrow i2[b_k])$. The actual encoding uses additional auxiliary literals to share common subexpressions. Again, we take a two-input AND-gate as an example. For better readability we just list the implications. The first part handles the cases with up to one X-value on the inputs:

$$\begin{array}{rcl} \neg i1[n] \wedge \neg i1[x] & \rightarrow & set\_o\_zero \\ \neg i2[n] \wedge \neg i2[x] & \rightarrow & set\_o\_zero \\ i1[n] \wedge \neg i1[x] \wedge i2[n] \wedge \neg i2[x] & \leftrightarrow & o[n] \wedge \neg o[x] \\ \neg o[n] \wedge \neg o[x] \wedge i1[n] \wedge \neg i1[x] & \rightarrow & \neg i2[n] \wedge \neg i2[x] \\ \neg o[n] \wedge \neg o[x] \wedge i2[n] \wedge \neg i2[x] & \rightarrow & \neg i1[n] \wedge \neg i1[x] \\ i1[x] \wedge i2[n] \wedge \neg i2[x] & \rightarrow & set\_i1\_o \\ i2[x] \wedge i1[n] \wedge \neg i1[x] & \rightarrow & set\_i2\_o \\ o[x] \wedge i1[n] \wedge \neg i1[x] & \rightarrow & set\_i2\_o \\ o[x] \wedge i2[n] \wedge \neg i2[x] & \rightarrow & set\_i1\_o \end{array}$$

If there are X-values on both inputs, the b-literals of both signals have to be compared. This is handled by the auxiliary literals $equalx\_i1\_i2$ and $equal\_i1\_i2$:

$$\begin{array}{rcl} equalx\_i1\_i2 \wedge (i1[n] \leftrightarrow i2[n]) & \rightarrow & set\_i1\_o \\ equalx\_i1\_i2 \wedge (i1[n] \oplus i2[n]) & \rightarrow & set\_o\_zero \\ \neg equal\_i1\_i2 \wedge i1[x] \wedge i2[x] & \rightarrow & set\_o\_xnew \end{array}$$

The remaining auxiliary literals do the following: (1) if $set\_o\_zero = 1$, then o will be set to logic-0, (2) if $set\_o\_xnew = 1$, then o will be set to the new X-value which may be introduced by this gate – this may happen if two different X-values reconverge, (3) if $set\_i1\_o = 1$, then $o = i1$, (4) if $set\_i2\_o = 1$, then $o = i2$, (5) $equal\_i1\_i2 = 1$ if and only if all b-literals of $i1$ and $i2$ are equal.

$$\begin{array}{rcl} set\_o\_zero & \rightarrow & \neg o[n] \wedge \neg o[x] \\ set\_o\_xnew & \rightarrow & \neg o[n] \wedge o[x] \wedge \\ & & \forall k \text{ set } b_k \text{ accordingly} \\ set\_i1\_o & \rightarrow & (i1[n] \leftrightarrow o[n]) \wedge o[x] \wedge \\ & & \forall k(i1[b_k] \leftrightarrow o[b_k]) \\ set\_i2\_o & \rightarrow & (i2[n] \leftrightarrow o[n]) \wedge o[x] \wedge \\ & & \forall k(i2[b_k] \leftrightarrow o[b_k]) \\ equal\_i1\_i2 & \leftrightarrow & \forall k(i1[b_k] \leftrightarrow i2[b_k]) \\ equalx\_i1\_i2 & \leftrightarrow & equal\_i1\_i2 \wedge i1[x] \wedge i2[x] \end{array}$$

The cases 1) and 2) result in a low number of clauses per gate. Obviously, case 3) is more complex, but the number of literals

only increases logarithmically with the number of X-values to be distinguished, i. e. if the number of X-values doubles, only one more b-literal is necessary. The number of clauses for the comparison of X-values only grows linearly with the number of b-literals. Therefore, with a reasonable amount of X-values the overall size of the CNF stays moderate.

D. Valid Difference and D-Chain in RSL ATPG

In SAT- or QBF-based ATPG algorithms, the search for a test pattern is sped up by introducing D-literals and D-chains which explicitly model propagation paths of the fault effect [14]. This restricts the solver to only consider assignments leading to a test pattern or to a faster abort if such assignments do not exist.

In two-value based ATPG, each D-literal d_s is logic-1 if and only if signal s in the fault-free (s^g) and faulty (s^f) circuit model have complementary logic values (i. e. $d_s = s^g \oplus s^f$). The same definition of D-literals is used in the accurate QBF-based approach of [12].

The D-literals are subsequently used to model the propagation paths of the fault effect with the help of D-chains. Classically (e. g. [14]), this D-chain starts at the faulty signal and leads to each reachable output.

For the incremental construction and solving of the SAT instances [17, 18] considering the outputs reachable from the fault one at a time, a *backward* directed D-chain is more suitable: The backward D-chain starts at a reachable output and leads backwards to the faulty signal, enforcing that a difference at the output (of a gate) implies a difference at at least one of its inputs. This allows to expand the SAT instance with reachable outputs and corresponding D-chain clauses incrementally.

For the example in Figure 3, the use of a backward D-chain leads to the implications $(d_4 \rightarrow (d_2 \vee d_3))$, $(d_3 \rightarrow d_1)$, $(d_2 \rightarrow d_1)$ as well as $d_4 = 1$ if only this output is considered. In case several outputs are considered, an additional clause is added to guarantee that at least one D-literal of an observable output shows logic-1.

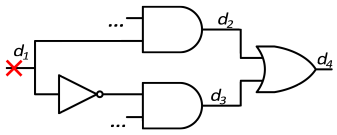


Fig. 3: Propagation of a fault located at d_1.

In the proposed restricted symbolic ATPG, the backward D-chain is constructed in a similar way. However, the semantic of the D-literals is different. As stated above, classically a valid difference at signal s, i. e. $d_s = 1$, always means that s has complementary binary values in the fault-free and faulty circuit. In the proposed RSL ATPG, the complementary binary values are only required at a the detecting output.

At internal signals in the propagation cone of a fault, the following three cases are considered a valid difference as well: (i) The signal carries X-values with different polarity in the fault-free and faulty circuit, i. e. X_i and $\neg X_i$. (ii) The signal carries an X-value either in the fault-free or faulty circuit but not in both. The X-value may reconverge later and cause complementary binary values at an output. (iii) The signal carries different X-values in the fault-free and faulty circuit, which may reconverge later and cause complementary binary values at an output.

It can be shown that with theses three cases propagation within RSL is completely modeled. Unfortunately, most of the possible assignments to the signals on the propagation path are therefore considered as a valid difference. Thus, the D-chain is less effective in pruning the search space for a test pattern and the solver has to explore many possible propagation paths. On the contrary, not considering these three cases might result in a loss of fault coverage, as valid propagation paths are excluded from consideration.

In the following, we explain how to optimize the construction of the D-chain without loosing accuracy: As explained in Section II-C, the n-literal denotes whether a signal value is negated or not. Hence, a logic-0 is represented as $n = 0, x = 0$, and a non-negated X-value implies $n = 0$. Furthermore, a visible binary difference at signal s implies that s shows complementary logic values in the fault-free and faulty circuit, i. e. the n-literals of s differ.

As an example, consider a two-input OR gate with one input showing a logic-0 ($n = 0$) in the fault-free and X_i ($n = 0$) in the faulty circuit. The output can only show a binary difference if the second input has logic-0 ($n = 0$) in the fault-free and $\neg X_i$ ($n = 1$) in the faulty circuit. For the other gate types, similar examples exist. They all have in common that the n-literals of one input in the fault-free and faulty circuit are equal, while the n-literals of the other input differ. Hence, for all gate types, X-canceling requires these two cases at the inputs. With the D-chain modeling introduced above in (ii) and (iii), both of these cases would be considered as difference. However, as we need both of them, it is sufficient to propagate only one in the D-chain since the solver will enforce the other case due to the gate modeling (cf. Section III-C).

Therefore, we constrain the D-literals such that only the cases with equal n-literals of a signal in the fault-free and faulty circuit are considered as a valid difference. This reasoning is valid for gates with more than two inputs as well.

A further restriction of valid differences leads to even tighter D-chains but may cause a (minor) loss of accuracy. Still, the tighter D-chains prune the search space considerably, which reduces the runtime. Therefore, in addition to the complete RSL ATPG, we propose an optimized RSL ATPG with notably reduced runtime. Experimental results will show that this can be done without compromising the overall quality.

E. Complete RSL ATPG

The complete RSL approach detects a fault if and only if it is detectable in restricted symbolic logic. Therefore, all possible differences as stated in Section III-D are considered by the D-literals and D-chains, as they may lead to a valid test pattern.

Only this complete RSL approach is able to prove the RSL-untestability of a fault by showing the unsatisfiability of the corresponding SAT instance. However, such a comprehensive D-chain may prevent significant pruning of the search space resulting in high runtimes.

F. Optimized RSL ATPG

The optimized RSL approach limits the differences which are considered in the D-literals: All combinations for which either the fault-free or faulty circuit model, but not both, show an X-value are forbidden in the D-chain since they incur high runtime and are rarely required for fault propagation. This implies much tighter constraints for the search of a test pattern. Consequently, not all faults detectable by restricted symbolic logic may be detected by this optimized approach. Nonetheless, this approach detects a superset of the faults detectable by three-valued ATPG. In Section V-A we compare the accuracy of the optimized with the complete RSL ATPG. In Section V-B, we combine the optimized RSL ATPG with accurate fault simulation [10] to compensate the potential loss in fault coverage compared to the complete approach. This fault simulation uses accurate Boolean modeling and allows to classify faults as detectable for which both optimized and complete RSL ATPG may not be able to generate a test pattern.

IV. ATPG Framework

The proposed restricted symbolic ATPG approaches have been extended with fault simulation and a topological untestability check [12] to minimize runtime. Figure 4 shows the resulting ATPG framework combining the proposed ATPG ③ with a hybrid two- and three-valued SAT-based ATPG ①, the topological untestability check ②, and fast fault simulation with high accuracy in presence of X-values ④.

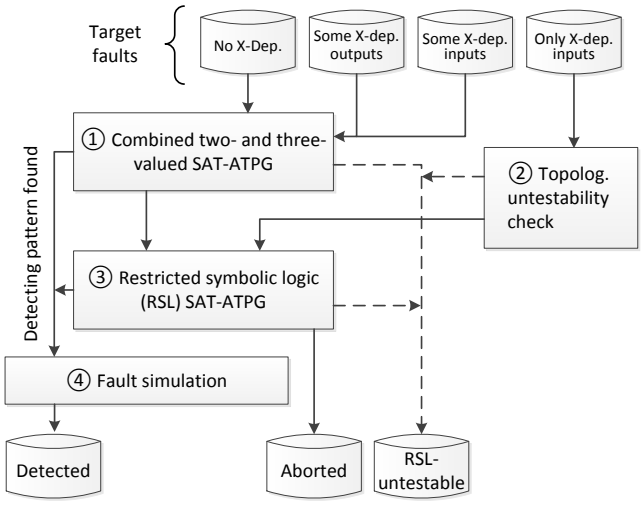


Fig. 4: Proposed ATPG framework

Prior to each ATPG run, the target faults are partitioned into four different groups. The first group (No X-dep.) contains faults that do not depend on the X-sources. Each of these faults can be analyzed accurately using the combined two- and three-valued SAT-based ATPG. If a test pattern is found, fault simulation with high accuracy (see below) is performed to implement fault dropping (i. e. all faults also detectable by this test pattern are marked as detected). Otherwise, the fault is marked as RSL-untestable.

After the faults of the first group are classified, all faults for which only a subset of the reachable outputs or a subset of the inputs in the adjustment cone depend on X-sources, are also analyzed by the two- and three-valued SAT-based ATPG. However, if the ATPG cannot find a test pattern for these faults, the faults are not necessarily untestable. Therefore, the undetected faults of this group are later analyzed using the proposed restricted symbolic ATPG. If restricted symbolic ATPG finds a test pattern, it is simulated for fault dropping. If no test pattern is found, the fault is marked as RSL-untestable.

The last group contains all faults whose input cone is exclusively driven by X-sources, i. e. fault activation only depends on X-sources. A binary value at the fault site can only be generated if X-values reconverge in the input cone. The *topological untestability check* traces fanout branches and checks whether reconvergences exist. If not, the fault is marked as untestable. Otherwise, the fault is analyzed by the RSL ATPG.

As the satisfiability check of a SAT instance is an NP-complete problem, a timeout may be used for practical reasons depending on the circuit size and complexity. All faults not classified within the timeout are marked as aborted.

Fault Simulation: Two different fault simulation strategies with much higher accuracy than classical algorithms are used in this work. The complete RSL approach is combined with a restricted symbolic logic based simulation engine to investigate the achievable fault coverage with restricted symbolic logic. In contrast, the optimized RSL ATPG uses the accurate SAT-based fault simulation of [10, 11] which computes the accurate fault coverage in presence of X-values by a mapping to a SAT-instance. As shown by the experimental results, this balances runtime and accuracy.

V. Evaluation

The proposed algorithms are implemented in C. All SAT-based approaches use the incremental SAT-solver antom [19]. For comparison, the accurate QBF-based ATPG of [15] is used. We evaluated the algorithms on full-scan circuits of the largest ISCAS'85 (c6288, c7552) and ISCAS'89 (cs38417) benchmarks as well as larger industrial designs from NXP. The experiments were conducted on an Intel Xeon CPU with 3.3 GHz. We assume that a fixed and randomly selected subset of circuit inputs generates X-values.[2] Five different subsets of X-source inputs are generated per circuit. The reported results are the rounded average over these five experiments per circuit. For each circuit, the collapsed set of stuck at faults is computed. 1024 random patterns are simulated for each set of X-sources. The remaining random-pattern resistant faults are processed using the proposed algorithms.

[2]The effect of clustered X-sources has been discussed in detail in [11].

A. Accuracy of ATPG Based on Different Logics

The first experiment investigates the attainable fault coverage depending on the underlying logic in test pattern generation. An ATPG based on three-valued logic is compared with the accurate QBF-based approach of [15], the proposed complete restricted symbolic logic (RSL) ATPG (Section III-E), as well as the optimized RSL ATPG (Section III-F). For the optimized approach, we disable accurate fault simulation to measure the loss of accuracy during ATPG due to the tightened D-chain modeling.

In this experiment, the timeout per fault used in [15] for the QBF-based ATPG is doubled (22 seconds) to classify as many faults accurately as possible.[3] For all other approaches, no faults are aborted.

Figure 5 shows the achieved fault coverage considering up to 20% of the inputs as X-sources for the ISCAS85 circuit c7552 – an adder with comparator and parity logic. Additionally, the absolute difference between the optimized and the complete RSL approach, and the absolute difference between the optimized RSL and the accurate approach is displayed.

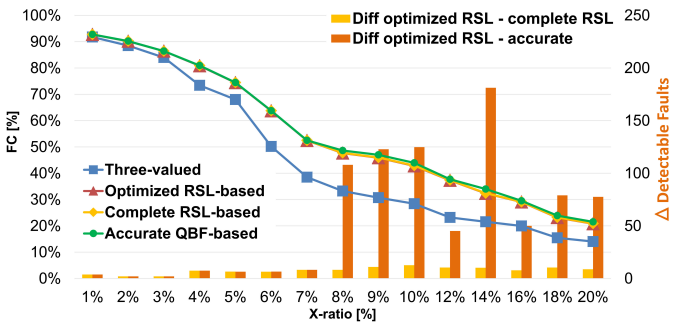


Fig. 5: Fault Coverage of ATPG based on different logics for circuit c7552.

ATPG based on three-valued logic already leads to a 1% loss in fault coverage compared to the more accurate solutions if only 1% of the inputs are selected as X-sources. Higher numbers of X-sources further increase the pessimism of three-valued logic. The greatest difference is observed when 12% of the inputs are selected as X-sources. Here, the fault coverage of the three-valued ATPG is over 37% lower than in the optimized RSL-based approach.[4]

Comparing the optimized RSL-based and the complete RSL-based approach, the fault coverage for all tested X-ratios is about the same. For X-ratios smaller than 5%, the difference is below 0.05% (i. e. six detectable faults). On average for circuit c7552, the optimized RSL-based approach leads to a loss of less than 0.09% compared to the complete RSL-based approach (i. e. seven faults). Similar results are achieved for the larger circuits in the experiments in the next section.

Comparing the results of the complete RSL-based approach to the accurate QBF-solution and X-ratios smaller than 5%, the difference in fault coverage is negligible and below 0.01% (one or no detectable fault). For higher X-ratios, the difference grows slightly. The peak is reached for an X-ratio of 14% with a difference of 4.6% (i.e. 181 detectable faults). For this X-ratio, ATPG based on complete RSL already detects 1 165 faults more than the three-valued approach (i.e. 50%).

However, as the number of X-sources grows, the number of faults not classified by the QBF-based approach within 22 seconds increases. On average, 4.60% of the faults were aborted. Hence, there are still some faults left, which might be detectable using the QBF-based ATPG with an even higher timeout. Nevertheless, the difference in fault coverage between both restricted symbolic approaches and the accurate solution is quite low.

B. Fault Coverage in Larger Circuits

In the second experiment we provide comparative results for several larger benchmark circuits. In particular, we consider the optimized RSL ATPG *with* the accurate fault simulation of [11] since this combination promises a high fault coverage and good scalability at the same time. In this way, it is possible to compensate the small loss in coverage due to the D-chain optimization without a negative effect on the overall runtime.

The achieved fault coverage is compared to the complete RSL ATPG, a three-valued ATPG, and the accurate QBF-based ATPG of [15]: The accurate QBF-based ATPG is used to demonstrate the quality achieved by RSL ATPG methods presented in this paper. On the other hand, three-valued ATPG provides data on how these faults are handled in classical algorithms showing the prevailing pessimism in widely used tools. Results of experiments with a state-of-the-art X-aware commercial ATPG are in line with the fault coverage of the used three-valued ATPG. These tools seem to be incapable of evaluating reconvergences of X-values accurately.

For all considered approaches, a timeout of 11 seconds per fault is used. All faults not classified within this timeout are marked as aborted. The runtime of the QBF-based approach for circuit p141k already exceeds 48h. For circuit p267k with 271 538 gates and over 650 000 faults, no results were available after 72h. Thus, no results are given for this circuit, and we do not list even larger circuits.

Table I shows the results in detail. For each circuit, the name, the number of gates and the number of collapsed stuck-at faults are listed in columns 1 to 3. Per circuit, we conduct the experiments for 1%, 2% and 5% of the circuit inputs as X-sources ('X-ratio' in column 4). For circuit c6288 with only 32 inputs, the case of 2% is omitted since an X-ratio of 1% and 2% results in a single X-source.

For each of these cases the fault coverage for a three-valued ATPG is given in column 5. The results of the proposed complete approach are given in columns 6 to 8 ('Complete RSL ATPG'). Columns 9 to 11 provide data for the optimized

[3] An even higher timeout of 60 seconds results only in a few more faults classified as untestable but still does not allow to classify all faults accurately.

[4] The difference in fault coverage is not monotonously falling as different X-ratios may lead to different numbers of reconvergences and therefore decreases or increases the pessimism of three-valued logic.

RSL ATPG including accurate simulation ('Optimized RSL ATPG & acc sim'). Columns 12 to 14 list the results of the accurate QBF-based approach.

The results show that both restricted symbolic logic based approaches are less pessimistic than three-valued ATPG and classify a higher number of faults as detectable. For an X-ratio of 5% and the ISCAS benchmarks, the fault coverage increases by up to 33% (from 70.47% to 94.29% for circuit c6288) using the complete RSL ATPG, or by up to 35% (from 70.47% to 95.04% for circuit c6288) when the optimized RSL ATPG is used.

For the industrial circuits from NXP, the complete RSL ATPG classifies up to 16 601 more faults as detectable than the three-valued ATPG (p78k), i. e. an increase in fault coverage of up to 8.7%. With the optimized RSL ATPG, the difference is even higher, and fault coverage increases by up to 9.2% (i. e. 17 424 faults).

In most cases, the fault coverage of both RSL-based approaches is also close to the accurate result. The maximum loss in accuracy of the complete RSL ATPG is measured for circuit c6288 and an X-ratio of 5%. Here, the achieved fault coverage is 0.80% lower than the accurate solution. For the optimized RSL ATPG including accurate fault simulation, the difference is even smaller. The highest loss in fault coverage of only 0.28% is measured for circuit p78k and an X-ratio of 5%. Hence, the combination of optimized RSL ATPG with accurate fault simulation allows in most cases to compensate the inaccuracy of RSL logic and thus, fault coverage deviates only slightly from the accurate result.

On the other hand, as the number of aborted faults of the QBF-based approach increases for larger circuits, both RSL-based approaches may also show a higher fault coverage than the QBF-based approach. Thus, for circuit p141k, the complete RSL ATPG classifies up to 0.27% more, and the optimized RSL ATPG classifies up to 0.30% more faults as detectable than the QBF-based approach.

On average over all circuits, the *complete RSL ATPG* achieves only a 0.07% lower fault coverage than the accurate solution,[5] and the fault coverage of the optimized RSL ATPG which includes accurate fault simulation is only 0.02% lower compared to the accurate solution. This shows that for almost all testable faults, the accuracy of RSL is sufficient to find a test pattern, and only very few faults require accurate reasoning.

The runtime of the complete RSL ATPG is on average one order of magnitude lower than the accurate QBF-based algorithm. The optimized RSL approach further decreases the runtime by on average two orders of magnitude compared to the QBF-based approach. The largest difference for ISCAS benchmarks of factor 618 is achieved for circuit c7552 and an X-ratio of 2%. For industrial circuits, the largest speed-up of $236\times$ is achieved for circuit p78k and an X-ratio of 5%. In this case, the runtime drops from over 7 hours to 113 seconds.

[5]This represents the limit of any pure RSL-based fault simulation and test generation approach.

The proposed RSL ATPG algorithms are also much more robust than the accurate QBF-ATPG: The number of aborted faults is significantly smaller. The complete RSL approach decreases the number of aborts by 54.8%, and the optimized RSL approach aborted no faults for all but one circuit (reduction by 92.1%).

These results demonstrate that RSL ATPG offers comparable quality to the accurate approach and scalability to industrial circuits at the same time.

VI. Conclusions

In this paper we presented novel SAT-based algorithms for test generation in presence of unknown values. Unknown values are modeled in the so-called restricted symbolic logic (RSL) known to provide superior accuracy compared to standard three-valued logic.

We propose the first *complete RSL ATPG algorithm*, i. e. the algorithm generates a test pattern if and only if the fault considered is testable in RSL logic. To do so, it includes an extended version of D-chains that explicitly models the propagation paths with the resolution of RSL. It achieves a fault coverage very close to the accurate result of a QBF-based ATPG, while being one order of magnitude faster and having fewer aborted faults.

To further reduce the runtime, we propose an *optimized RSL ATPG* based on a more restricted version of D-chains. This reduces the number of fault propagation paths considered during the test pattern search. To compensate the minor loss of accuracy, the algorithm is combined with accurate fault simulation. Experimental results show the high accuracy, scalability, and robustness of this optimized approach. The achieved fault coverage in many cases even exceeds that of the complete approach. On average, the optimized RSL ATPG is two orders of magnitude faster than the QBF-based accurate solution. In addition, aborted faults are reduced by 92.1%.

Acknowledgments

The authors thank Linus Feiten, Tobias Schubert and Sven Reimer from the University of Freiburg and Sudhakar Reddy from the University of Iowa for supporting this work. This work was partially supported by the German Research Foundation (DFG) under grants BE 1176/14-2, SFB/TR14 AVACS, WU 245/9-1 (INTESYS), WU 245/11-1 (OASIS), and GRK1103.

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] P. Muth, "A nine-valued circuit model for test generation," *IEEE Trans. on Computers*, vol. C-25, no. 6, pp. 630–636, june 1976.

[3] A. Jain, V. Boppana *et al.*, "Testing, verification, and diagnosis in the presence of unknowns," in *Proc. IEEE VLSI Test Symposium (VTS)*, 2000, pp. 263–268.

[4] S. B. Akers, "A logic system for fault test generation," *IEEE Transactions on Computers*, vol. C-25, no. 6, pp. 620–630, june 1976.

[5] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[6] J. Carter, B. Rosen *et al.*, "Restricted symbolic evaluation is fast and useful," in *Proc. IEEE International Conference on Computer-Aided Design (ICCAD)*, 1989, pp. 38 –41.

[7] M. A. Breuer, "A note on three-valued logic simulation," *IEEE Transactions on Computers*, vol. 21, no. 4, pp. 399–402, Apr. 1972.

[8] T. Ogihara, S. Saruyama, and S. Murai, "Test generation for sequential circuits using individual initial value propagation," in *Proc. IEEE International Conference on Computer-Aided Design (ICCAD)*, 1988, pp. 242–247.

[9] S. Kundu, I. Nair *et al.*, "Symbolic implication in test generation," in *Proc. Conference on European Design Automation*, 1991, pp. 492–496.

[10] S. Hillebrecht, M. A. Kochte *et al.*, "Exact stuck-at fault classification in presence of unknowns," in *Proc. IEEE European Test Symposium (ETS)*, 2012, pp. 1–6.

[11] 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.

[12] S. Hillebrecht, M. A. Kochte *et al.*, "Accurate QBF-based test pattern generation in presence of unknown values," in *Proc. Conf. on Design, Automation and Test in Europe (DATE)*, 2013, pp. 436–441.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979.

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

[15] D. Erb, M. A. Kochte *et al.*, "Accurate multi-cycle ATPG in presence of X-values," in *Proc. IEEE Asian Test Symposium (ATS)*, Nov 2013, pp. 245–250.

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

[17] N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," *Electr. Notes Theor. Comput. Sci.*, vol. 89, no. 4, pp. 543–560, 2003.

[18] D. Tille and R. Drechsler, "Incremental SAT instance generation for SAT-based ATPG," in *11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, April 2008, pp. 1–6.

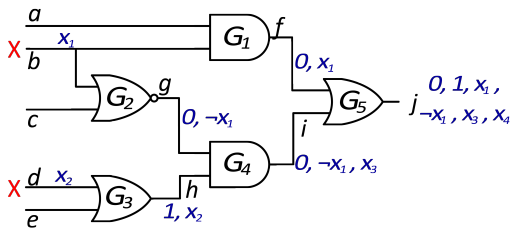
[19] T. Schubert, M. Lewis, and B. Becker, "Antom—solver description," *SAT Race*, 2010.

TABLE I: RESULTS OF THE PROPOSED ATPG IN CONTRAST TO A THREE-VALUED AND ACCURATE SOLUTION.

circuit	gates	faults	X-Ratio	3-val. ATPG	**Complete RSL**			**Optimized RSL & acc. sim.**			Accurate ATPG of [15]		
			[%]	FC[%]	FC[%]	Aborts	Time[s]	FC[%]	Aborts	Time[s]	FC[%]	Aborts	Time[s]
c6288	2 416	8 704	1.0	86.43	97.38	3	50	97.38	0	12	97.38	9	110
			5.0	70.47	94.29	41	676	95.04	0	22	95.04	25	296
c7552	4 043	10 816	1.0	91.80	92.77	0	2	92.74	0	2	92.77	91	1 089
			2.0	88.51	90.23	0	4	90.21	0	3	90.23	169	2 168
			5.0	68.02	74.50	0	15	74.44	0	10	74.50	254	3 478
cs38417	23 537	59 041	1.0	95.54	95.71	0	25	95.71	0	17	95.71	32	439
			2.0	93.58	93.83	0	33	93.83	0	21	93.83	57	799
			5.0	86.54	86.99	0	64	86.99	0	32	86.99	86	1 085
p78k	74 243	225 476	1.0	97.30	98.67	0	282	98.67	0	17	98.69	241	3 656
			2.0	93.60	96.97	0	639	96.94	0	35	97.03	506	7 506
			5.0	84.25	91.61	2	2 099	91.97	0	113	92.23	1 762	26 989
p89k	88 726	239 090	1.0	91.11	92.11	0	2 303	92.11	0	1 578	92.11	1 364	22 693
			2.0	85.49	86.59	0	3 119	86.59	0	1 924	86.59	1 430	27 992
			5.0	70.90	72.76	11	4 293	72.76	0	2 789	72.76	1 726	32 942
p100k	96 685	259 322	1.0	95.31	96.16	432	8 069	96.12	15	2 080	96.16	979	15 473
			2.0	91.48	93.48	646	12 519	93.45	15	2 565	93.54	1 169	20 404
			5.0	80.54	83.75	4 421	60 947	83.80	1 597	24 745	83.99	3 867	65 154
p141k	172 686	452 599	1.0	95.18	96.24	630	13 546	96.24	0	3 645	96.22	810	43 303
			2.0	93.25	94.51	941	19 467	94.51	0	4 754	94.45	1 410	68 106
			5.0	85.58	87.61	2 171	43 045	87.63	0	10 598	87.37	4 532	182 282
p267k	271 538	658 395	1.0	94.71	95.06	41	13 934	95.02	0	11 724	-	-	-
			2.0	90.95	91.38	132	18 535	91.33	0	15 060	-	-	-
			5.0	80.00	81.99	218	28 647	81.97	0	24 944	-	-	-

APPENDIX

The example below shows a circuit with two X-sources X_1, X_2 as well as the information gained by the preprocessing explained in Section III-B. Additionally, the table below lists for each signal the resulting n-, x- and b-literals (columns 2 to 4) according to Section II-C. If only one gate input is X-dependent, no additional b-literals are generated. Furthermore, column 5 lists all assignments to literals already determined by the preprocessing. For this circuit, 25 variables are sufficient for the signal encoding with RSL resolution. To test the stuck-at-1 fault at signal j, it must be set to logic-1. In the proposed encoding, this is achieved by assigning $l_{21} = 1, l_{22} = 0$.



signal	n-literal	x-literal	b-literals	Comments
a	$a[n] = l_1$			
b	$b[n] = l_2$	$b[x] = l_3$	$b[b_1] = l_4$	Signal b always shows X_1 leading to the assignments: $l_2 = 0, l_3 = 1, l_4 = 1$
c	$c[n] = l_5$			
d	$d[n] = l_6$	$d[x] = l_7$	$d[b_1] = l_8$ $d[b_2] = l_9$	Signal d always shows X_2 leading to the assignments: $l_6 = 0, l_7 = 1, l_8 = 0, l_9 = 1$
e	$e[n] = l_{10}$			
f	$f[n] = l_{11}$	$f[x] = l_{12}$		Signal f may only show a logic value or the same X-value as signal a. Therefore no b-literals are necessary.
g	$g[n] = l_{13}$	$g[x] = l_{14}$		At signal g, as for signal f, no b-literals are required.
h	$h[n] = l_{15}$	$h[x] = l_{16}$		Signal h may only show logic-1 or the X-value of signal d. Hence, no b-literals are required.
i	$i[n] = l_{17}$	$i[x] = l_{18}$	$i[b_1] = l_{19}$ $i[b_2] = l_{20}$	At signal i, the b-literals may only represent X_1 or X_3. This leads to: $l_{19} = 1$
j	$j[n] = l_{21}$	$j[x] = l_{22}$	$j[b_1] = l_{23}$ $j[b_2] = l_{24}$ $j[b_3] = l_{25}$	In order to adjust a logic value, no X-values are allowed for signal j.

Fig. 6: Restricted symbolic literal encoding (according to Section II-C).