

Decision Selection and Learning for an ‘All-Solutions ATPG Engine’ *

Kameshwar Chandrasekar and Michael S. Hsiao (*{kamesh, hsiao}@vt.edu*)
Department of Electrical and Computer Engineering, Virginia Tech
Blacksburg, VA, 24061

Abstract

‘All-solutions ATPG’ based methods have found applications in Model Checking sequential circuits, and they can also improve the defect coverage of a test-suite, by generating distinct multiple-detect patterns. Conventional decision selection heuristics and learning techniques for an ATPG engine were originally developed to ‘quickly’ find any available (single) solution. Such decision selection heuristics may not be the best for an ‘all-solutions ATPG’ engine, where all the solutions need to be found. In this paper, we explore new techniques to guide an ‘all-solutions ATPG engine’. We first present a new decision selection heuristic that makes use of the ‘connectivity of gates’ in the circuit in order to obtain a compact solution-set. Next, we analyze the ‘symmetry in search-states’ that was exploited in ‘Success-Driven Learning’ [1] and extend it to prune conflict subspaces as well. Finally, we propose a new metric that determines the use of learnt information a priori. This information is stored and used efficiently during ‘success driven learning’. Experimental results show that we can compute the complete solution-set with our new heuristics for large ISCAS ’89 and ITC ’99 circuits, where conventional guidance heuristics fail.

1 Introduction

In recent years, Automatic Test Pattern Generation (ATPG) / Boolean Satisfiability (SAT) based methods have offered a *potential substitute* for Reduced Ordered Binary Decision Diagrams (ROBDD) based methods, to the verification community [1–7]. Unlike ROBDD based methods that can suffer from memory explosion, ATPG/SAT based methods can perform image or preimage computation with reduced memory requirements. Image/Preimage computation is performed by modifying the underlying ATPG algorithm, to generate all the available solutions, and it is a key step in sequential equivalence checking and unbounded model checking. In addition to design verification, ATPG engines that are able to generate multiple solutions can also be used to generate different and distinct multiple-detect test vectors for a given fault, thus improving the overall defect coverage of a test suite.

Traditionally, ATPG engines are guided by the testability measures of the circuit. Several heuristics have been developed to determine these measures [8] that help to find *any* available solution quickly. The distance based testability measures account for the difficulty of testing a gate, based on its distance from the primary inputs and primary outputs. In [9], the authors compute the 0/1 probability at the output of each gate in the circuit and use them as testability measures. In [10], the authors derive certain numbers called SCOAP measures, for each gate in the circuit, that represent the difficulty of justifying and propagating a value. All these testability measures guide the ATPG engine while *backtracking* from the objective to select a decision variable. Improvements to obtain better testability measures have been incorporated in [11] using the concept of super-gates. However, the worst-case complexity of obtaining these measures can be exponential. Recently, in [12], Chang et al. obtained a better approximation for the testability measures using implications generated in the circuit. Based on implication reasoning, they estimate a correlation factor that accounts for signal correlations in the circuit. However, all these testability measures aim at finding a *single* solution quickly. Likewise, in SAT, many variable/decision selection strategies have been proposed, such as MOMS (Maximum Occurrences in clause of Minimum Size), DLIS (Dynamic Largest Individual Sum) [13] and VSIDS (Variable State Independent Decaying Sum) [14]. However, these methods lack the structural information available to ATPG engines. In [15], Iyer et al. integrated SAT and ATPG to develop an ATPG based SAT solver. They use the variable selection strategy of ATPG due to their superiority in choosing variables related to the objective.

An *all-solutions ATPG* attempts to build a complete decision tree (which can be reduced to a graph by sharing common sub-trees) that is essentially a Free Binary Decision Diagram (FBDD). Conventionally, the testability measures of the circuit guide an ATPG engine to select decisions as they search for the solution. As a result, the variable order in the Free BDD conform to the testability measures of the circuit. It is necessary to bias the variable order in such a way that we obtain a compact Free BDD as a whole. A compact Free BDD helps to reduce the size of the decision tree and in turn speeds up the ATPG engine. If the number of solutions is very large, in the order of billions, then it is not possi-

*supported in part by NSF Grants CCR-0196470 and CCR-0305881.

ble to store each solution one-by-one due to memory and time limitation. This phenomenon is referred to as *solution explosion* in [1] and it was shown that the final solution-set can be efficiently represented by the decision tree as a Free BDD. Therefore, it is necessary to obtain a compact Free BDD in order to address the solution explosion problem as well. Furthermore, as each ATPG decision leads to a different search-state in the decision tree, the number of search-states can be *exponential* in the number of inputs. Since each search-state is stored in a hash-table, for use in success-driven learning [1], the memory required to store all the search-states becomes a critical issue. Storing all the search-states in a knowledge-base may potentially lead to memory explosion for large circuits. In order to reduce the size of the *knowledge-base* and still benefit from useful search-states, it may be sufficient to store only the *frequently* occurring search-states.

Given the above discussions, the contribution of this paper is three-fold:

1. We propose a new *decision selection heuristic* that guides an ‘all-solutions ATPG engine’ to obtain an efficient variable order for the Free BDD.
2. We introduce the concept of *symmetry in search states* for ATPG and analyze a theoretical formulation for *success driven learning* proposed in [1].
3. In order to reduce the number of search-states stored during success driven learning, we propose a new *metric* that determines the *use* of a search-state.

The rest of the paper is organized as follows. We introduce the background on variable ordering and learning in section 2. The new *decision selection heuristic* for an ‘all-solutions ATPG engine’ is presented in section 3. In section 4, we introduce ‘symmetry in search states’ for ATPG and show that success driven learning is a restricted realization of search-state based symmetry. A new metric developed to determine the use of a search-state is introduced in section 5. Section 6 presents the experimental results and section 7 concludes the paper.

2 Related Work on BDD Variable Ordering and Learning Heuristics

OBDD based methods are very sensitive to their variable order and thus are limited to small and medium sized circuits. Significant amount of work has been done on finding an efficient variable ordering technique in [16–21]. Most of the work aim at placing *related* variables together in order to obtain a compact BDD. In [19], a PODEM based variable ordering technique is considered for building ROBDDs. The testability measures of PODEM are used to backtrack to the primary inputs by a depth-first search and the inputs connected by shorter paths are placed together. Recently, Aloul et al. proposed static variable ordering techniques in [20, 21], and their experimental results showed that their techniques can be better than dynamic variable ordering techniques. They conjectured that placing *connected*

variables together and partitioning the variables lead to compact BDDs and faster SAT. On the other hand, Free BDDs (FBDDs) are relaxed versions of Ordered BDDs, in which variables can appear in different orders along different paths but each variable occurs only once along any given path. They are more compact than Ordered BDDs and sometimes lead to exponential savings in memory as shown in [22]. Although significant amount of work has been done to develop good variable ordering heuristics for ROBDDs, not much work has been done for Free BDDs.

In addition to variable selection heuristics, *learning* plays an important role in SAT/ATPG based methods. It helps to overcome the inherent time limitation of these methods and compete with BDD based methods. In [24–29], powerful learning techniques were introduced for ATPG. In [13, 14], efficient conflict-driven learning techniques were introduced for SAT based methods. In [3–5], learning techniques have been proposed for an *all-solutions SAT solver*. These learning techniques improve the efficiency of the SAT solver that is an integral part of unbounded model checking. In [6, 15], Iyer et al. combine the strengths of both SAT & ATPG and present efficient learning techniques for the *sequential justification* problem. In [7], Lu et al. proposed signal correlation guided learning for an ATPG based SAT solver and obtained a speedup for hard industrial circuits. In all these aforementioned ATPG/SAT engines, the knowledge is in the form of implications [24], assertions [27] or conflict clauses [13, 14, 29]. Efficient manipulation of knowledge is required to reduce the overhead in storing and using the knowledge base.

Recently, in [1], Sheng and Hsiao introduced a new type of ‘success-driven learning’ that efficiently prunes the search-space for ‘ATPG based preimage computation’, by identifying identical solution-subspaces. The Transition Relation is represented by a leveled circuit and the set of states is stored in a Free BDD. A PODEM based ATPG engine is invoked to find all the solutions, resulting in a preimage where all the current state variables are quantified. *Equivalent search-states* that lead to the same solution subspace are identified to prune the search-space. The decision tree, obtained during solution-search, is stored as a Free BDD that represents the complete preimage set. As solution subspaces heavily overlap during preimage computation, considerable savings is obtained in terms of time and memory. In [30], ‘augmented success driven learning’ and ‘search-state based conflict driven learning’ were introduced to further prune the search space for ATPG based preimage computation. However, the work in [1] and [30] used conventional testability measures to guide the ATPG engine, resulting in suboptimal all-solutions FBDDs.

2.1 Background on success-driven learning

Because an ATPG engine implicitly explores the entire search-space to generate a solution, ‘an all-solutions ATPG engine’ must continue and search for the next solution after each solution is found, until all solutions have been found. Each decision is considered a node in the decision tree, and

the entire search-space is explored to find all solutions. A few terms are introduced before explaining the concept of *success-driven learning* [1].

- **Decision Tree:** The tree obtained by the branch-and-bound procedure of ATPG, with input assignments as internal decision nodes, is called the decision tree.
- **Search-State:** After choosing each decision and performing logic simulation, logic values of all the internal gates form a state in the circuit. This internal state of the circuit after each decision is considered a *search-state* for the decision tree.
- **Cut-set:** Consider the circuit as a directed acyclic graph, C , with edges directed from primary inputs to primary outputs. If we remove the fanout-stems of a set of gates from C to partition the graph into two sub-graphs X and Y , such that all the edges (that exist in C) across X and Y are directed from primary inputs to primary outputs, then the set of gates is called a cut-set.
- **Cut-set for search-state:** Each search-state can be uniquely represented by a cut-set in the circuit. After each decision, the cut-set can be obtained by a multiple backtrace from the ATPG objective. The first frontier of *specified nodes*, encountered during backtrace, is the *cut-set for search-state*. In the sequel, we use the term cut-set to refer to “cut-set for search-state”. Cut-sets that lead to solution subspaces and their decision tree nodes are stored in a hash-table.

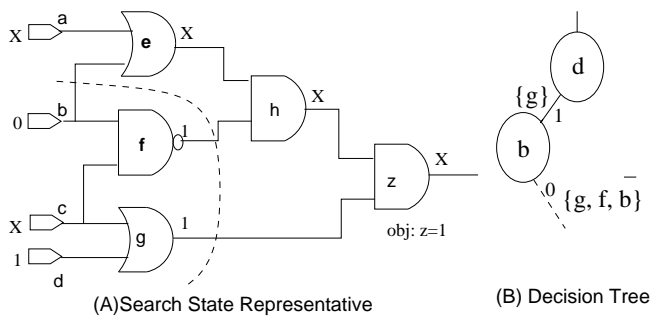


Figure 1. Cut-sets in the Search Space.

For the circuit in Figure 1(A), a partial decision tree is shown in Figure 1(B). The sets indicated on the decision edges (eg. $\{g\}$ and $\{g, f, \bar{b}\}$) are cut-sets for the corresponding search-states. For the last decision ($b = 0$), cut-set $\{g, f, \bar{b}\}$ is denoted by the dashed line, in Figure 1(A).

- **solution/conflict branch:** A branch in the decision tree that has at least-one/no solution below it.
- **solution/conflict cut-set:** A cut-set for the search-state in the *solution/conflict* branch.
- **solution/conflict subspace:** A search subspace below a *solution/conflict* branch.

In [1], the authors stored the cut-sets that lead to solution subspaces in a hash-table. A particular solution cut-set in

the circuit will lead to a specific solution subspace in the decision tree. If the same cut-set is encountered again, then we can simply link to that portion of the decision tree, instead of re-searching the same search space. After each decision, we search for the current cut-set in the hash-table. If an equivalent cut-set exists, we simply link the current branch to the stored node of the decision tree. Otherwise, we proceed with the usual search process. In this way, previously encountered search subspaces are not repeatedly explored using this technique.

3 New Decision Selection Heuristic

With the help of existing testability measures, conventional ATPG engines backtrack only through the easy (highly testable) portions of the circuit and stop as soon as *one* solution is found. Usually, this solution can be considered as the easiest solution with respect to the ATPG search process. However, an ‘all-solutions ATPG engine’ needs to explore the entire circuit to find *all* the available solutions. So the guidance heuristic for an all-solutions ATPG engine need not necessarily depend on the measures that only focus on the easier portions of the circuit.

We view the problem of finding all solutions that satisfy an objective as the process of constructing the decision tree as quickly as possible. This in turn is the problem of finding an efficient variable ordering technique for a Free BDD. The variable ordering heuristic is integrated into the backtrack routine of the ATPG engine. Unlike the dynamic variable ordering techniques in BDDs, we do not try to move the variable *up or down* to find a suitable position for a variable. Instead, after choosing each variable, we dynamically choose the next variable that is suitable for that position.

In [20], a hypergraph is built from the CNF formula and a *static variable order* is obtained by partitioning the variables and placing connected variables together. In order to obtain a good variable order, it is conjectured that *well connected* variables should be placed together for SAT and BDD. In our technique, we exploit the inherent graph structure of the circuit to dynamically find the gates that are well connected (by combinational paths) to the previously made decisions (represented by search-state in the circuit). After each decision, we estimate the *connectivity* of each gate to the previous decisions. While backtracing, we always choose the gate that has the highest connectivity measure. In some cases, we only consider the connectivity of a gate to the objective. In this way, we attempt to put all connected variables together in the decision tree. Gates that are well connected are likely to be highly correlated and hence should be chosen together. In a PODEM based ATPG, we decide on the primary inputs that are obtained by the backtrace routine. The following subsections further explain the concept of connected variables.

3.1 Connected Variables

At each decision during ATPG, the connectivity of a gate is estimated as the *number of combinational paths* that con-

nect the gate to the cut-set (formed by previous decisions) and the objective. From Graph Theory, a well-known linear time algorithm is sufficient to compute the number of paths passing through each gate in the circuit. A similar algorithm was used in [31] to estimate the fault coverage of path delay faults. Due to its linear time complexity, the computation overhead in estimating the connectivity measures is usually very small and it can easily be integrated into the backtrace routine of the ATPG engine.

```

// # paths through a gate = # paths in its fanin_cone *
// # paths in its fanout_cone
// # paths in fanin/fanout cone of each gate is counted once
function guidance_measures(levelized_ckt) {
  Initialize the dyn_msr of all gates to 0
  // step1. Initialize the dyn_msr of gates in the cutset
  for (each gate in the cut_set)
    dyn_msr[gate] = stat_msr[gate];

  // step 2. Update the dyn_msr for all gates in
  // the fanout_cone of the cutset
  for (each gate in fanout_cone of cut_set)
    dyn_msr[gate] = Sum(dyn_msr[fanins]) *
                    #paths[fanout_cone]
}

```

Figure 2. Update Dyn-Conn Measures.

Initially, the number of paths that connect a gate to the objective and the primary inputs is estimated. This measure is estimated for all gates in the circuit and stored as a *static connectivity measure*. After each decision, a *dynamic connectivity measure* is assigned for all gates in the fanout cone of the cut-set. A basic algorithm is shown in Figure 2. The gates in the cut-set are initialized to their static measures. These gates represent the previously made decisions in the decision tree. The gates outside the cut-set fanin/fanout cone (R3 in Figure 3) are not *directly* connected to the decisions made so far. So they are initialized to 0. Next, for each gate in the fanout-cone of the cut-set (R2 in Figure 3), the number of paths (dynamic connectivity measure), it occurs in, is recursively estimated from cut-set to the objective. Note that the paths due to the gates in R3 are ignored. In this way, gates that connect the cut-set to the objective are assigned dynamic connectivity measures depending on their connectivity to the objective and cut-set.

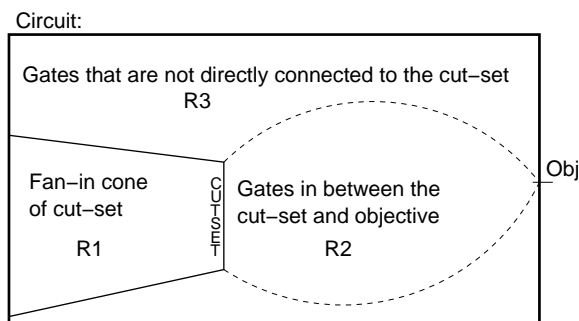


Figure 3. Dynamic & Static connectivity.

While backtracing from the objective, we will encounter two types of gates, as illustrated in Figure 3:

- Gates in between the cut-set and the objective - R2:** For these gates, we make use of the dynamic connectivity measures, since it is a representative of the connectivity of the gate to previous decisions and objective.
- Gates that are not directly connected to cut-set - R3:** For these gates, we use the static connectivity measures to select the gate that is well connected to the objective. Note that the dynamic connectivity measures for these gates are '0', because they are not directly connected to the cut-set.

It may be noted that if a gate, g , does not lie in between the cut-set and objective, i.e. $g \in R3$, then all the gates in the fanin-cone of g cannot lie in between the cut-set and objective, as shown in Figure 3. While backtracing, once we reach a gate with 0-dynamic connectivity measure we can start using the static connectivity measure for all the gates in its fanin cone. This helps us to implement an easy single switch from dynamic connectivity measures to static connectivity measures while backtracing in the circuit.

As discussed previously in Section 2.1, a cut-set is obtained by a multiple backtrace from the objective to the primary inputs of the circuit. Since we are choosing connected variables together, all elements in the cut-set tend to be closer to each other. As a result, we are more likely to obtain cut-sets with smaller widths.

3.2 An example

A slight variation of our technique is to select the variables based on static connectivity measures alone. In that case, we need not update the dynamic connectivity measures for every backtrace. Due to its ease of explanation, we use the static connectivity measures in this example. Figure 4 demonstrates our technique for a reconvergent structure that is present in many circuits. For the circuit shown in Figure 4(A), the objective of the ATPG engine is to find all solutions that satisfy the objective $h = 1$. The SCOAP measures, (C1, C0) and our static connectivity based measures, (m) are tabulated and listed in Figure 4(D). The traditional guidance heuristic in PODEM traverses the path $h - f - a$ and picks a as the first decision. Note that there was a choice at gate f on the path in which either a or e could be selected. a is chosen by SCOAP since it is easier-to-control when compared to e . This decision process continues and the final complete decision tree obtained for all solutions is shown in Figure 4(B). In this tree, a solid edge indicates the 1-branch and a dashed edge represents the 0-branch for each node.

For the same circuit in Figure 4(A), our new heuristic backtraces through e (after $h - f$) first, since it occurs in many paths as compared to a . If we follow our guidance heuristics for the rest of the backtraces, the corresponding decision tree obtained is shown in Figure 4(C). The first solution obtained by each technique is highlighted by a dotted

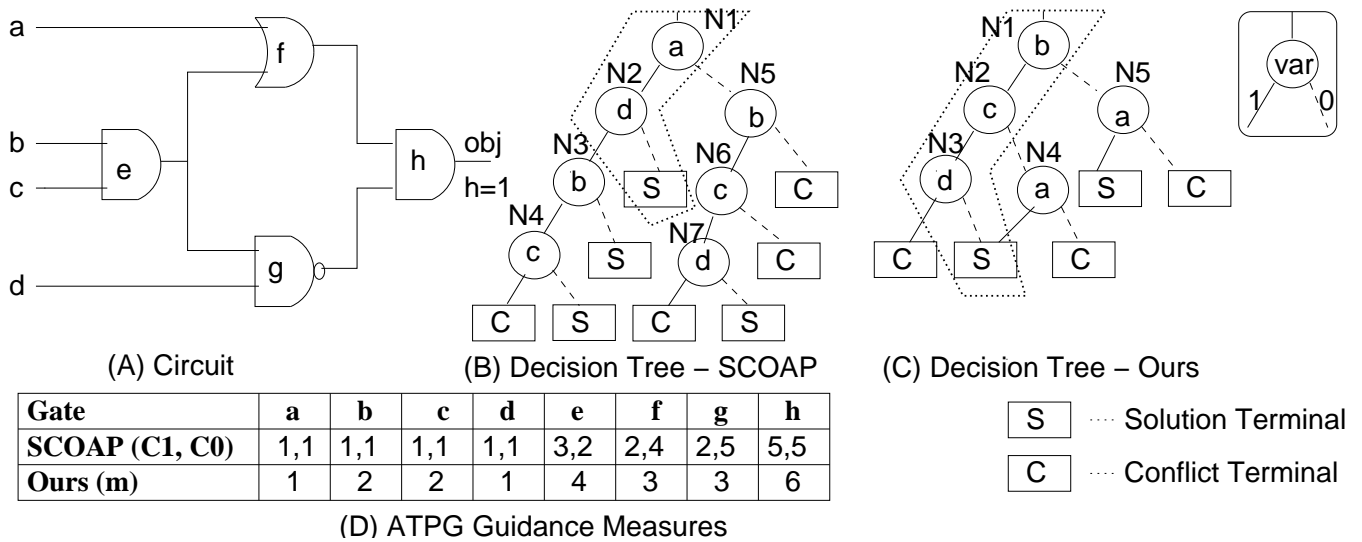


Figure 4. An Example to Illustrate the Effect of Different Testability Measures.

enclosure in the corresponding decision tree. It is seen that SCOAP-guided ATPG finds the first solution in only 2 decisions - $\{a, \bar{d}\}$, and our technique finds the first solution in 3 decisions - $\{b, c, \bar{d}\}$. However, the number of nodes in the decision tree 4(C) obtained by our technique (5) is less than the number of nodes in the decision tree obtained by using SCOAP measures (7). It may be noted that the nodes, N4 and N5 in Figure 4 (C) are identical. These nodes will be shared in the Free BDD and the effective number of nodes required to store the solution-set is only 4. Through this example, we see that although SCOAP finds the first solution in fewer decisions, our technique computes the complete solution-set in a fewer number of decisions.

4 Search-state based symmetry

In this section, we introduce a few definitions and theorems that help to analyze the search-states occurring in the decision tree of an ATPG engine.

Equivalence: Two search-states are said to be equivalent if they lead to the same sub-decision tree for a given ATPG.

Symmetry: Two partial input assignments are said to be symmetric if they form equivalent search-states.

For example, in Figure 4 (C) the search-states at nodes N4 and N5 are equivalent and the corresponding partial input assignments, $\{b, \bar{c}\}$ and $\{\bar{b}\}$ are symmetric. Note that this notion of symmetry is different from the ones that are generally used. Unlike previous methods, where permutations of fully specified input assignments and symmetry on two variables were used, we define symmetry on partial input assignments that form different decompositions of the circuit during ATPG.

Theorem 1 *If two cut-sets are equal (same), then they represent equivalent search-states.*

Proof: Cut-sets that are equal decompose a circuit to identical sub-circuits. If the same decision process is used, then the decision tree for identical sub-circuits will definitely be isomorphic. Since the sub-decision trees are the same, the corresponding search-states represented by equal cut-sets are equivalent. \diamond

Theorem 2 *All equivalent search-states are NOT necessarily represented by the same cut-set.*

Proof: We prove Theorem 2 by showing a counter-example, where two different (unequal) cut-sets represent equivalent search-states. Figure 5 (B) shows the ISCAS '89 circuit - s27, modified to find the one-cycle preimage for 001 at the next state flip-flops. The objective is to justify a 1 at the output of gate 22 in the circuit. The guidance measures are shown in brackets near each gate in the circuit. Note that the guidance measures are only heuristics, and they change the structure of the decision tree. Different guidance measures lead to different cut-sets. We show that for a fixed guidance heuristic, different cut-sets can sometimes lead to the same sub-decision tree.

A partial decision tree obtained during the ATPG is shown in Figure 5 (A). The solid edge is the 1-branch and the dashed edge is the 0-branch for each node. The nodes are labeled in chronological order of decisions. The terminal nodes refer to a solution or a conflict or an equivalent search-state (due to equal cut-sets) that was stored earlier. The link for the equivalent search-state terminals are shown by the node numbers next to the terminal nodes. For example, the 1-branch of N5 is connected to N4, since the corresponding cut-sets are equal. It may be observed in the decision tree that the 0-branch of N2 and 1-branch of N6 have isomorphic sub-trees below them. However, the corresponding cut-sets are unequal. By definition of equivalence, the two search-states represented by cut-sets - $\{19, \bar{10}\}$ and

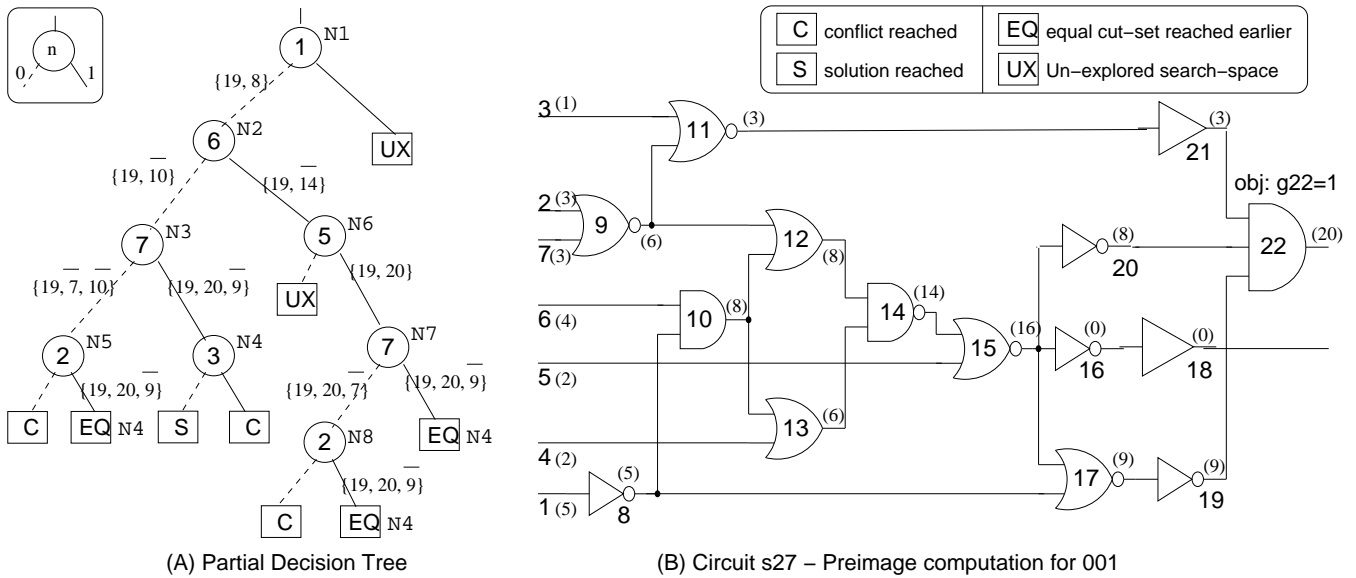


Figure 5. Counter-example for Theorem 2.

{19, 20} are equivalent. However, the corresponding cut-sets are different. ◊

Corollary Success driven learning is a restricted realization of equivalence in search-states.

Proof: In success driven learning, we use the cut-sets as a representative for the search-states. A cut-set is stored only if it leads to at least one solution. If an *exact* match for the solution cut-set occurs again, then we use the cut-set. From Theorem 2, we saw that unequal cut-sets can also represent equivalent search-states. On the other hand, even if a search-state leads to a conflict sub-space then its equivalent search-state will also lead to a conflict sub-space. This phenomenon is not exploited in success driven learning. ◊

From the above discussion, it is seen that certain cut-sets exist that lead to the same search sub-space and are not identified by success driven learning. Since equivalent cut-sets lead to the same search sub-space in general, it is immaterial if they are solution cut-sets or conflict cut-sets. As a result, success driven learning can be extended to prune conflict subspaces as well.

5 Cut-set Occurring Probability

Cut-sets that occur frequently during ATPG help to prune search subspaces, while other cut-sets may be less useful to the ‘all-solutions ATPG engine’. It is desirable to store only the cut-sets that occur frequently during the search process. Cut-sets that do not occur again or with low probability of occurrence can be ignored to save memory without loss of performance. Therefore, it prompted us to develop a metric that determines the usefulness of a cut-set. Based on this metric, we may decide to store a cut-set or ignore it. This will help to reduce the memory requirement for the knowledge base, without significant loss of information.

A cut-set, C , comprises of a set of specified gates, say

$$C = \{g_1, g_2, g_3, \dots, g_n\}$$

where, g_i is a gate at the frontier of the search-state/cut-set and, n is the number of gates in the cut-set.

A cut-set can occur multiple times only if it has many symmetric input assignments. An appropriate way to analyze the use of a cut-set is to count the number of symmetries that generate the same cut-set. If there are many symmetric assignments, then the cut-set is likely to occur again. For example, in Figure 6, the symmetric input assignments that generate the same cut-set $\{g, f, \bar{b}\}$ for the circuit are listed. It can be observed that it is very time-consuming to count the number of symmetric assignments that generate the same cut-set after each decision. Even if we count all the symmetric input assignments, a given ATPG engine may not consider all of them. As an alternative, we *estimate the use* of a cut-set using probabilistic measures.

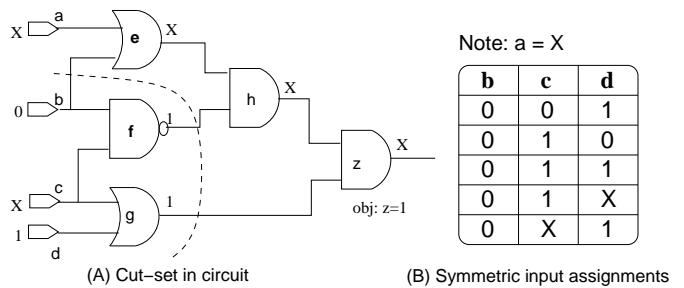


Figure 6. Symmetric Input Assignments.

The probability of occurrence of a 0 or 1 at the primary inputs is initialized to be 0.5. Then, we recursively estimate the probability of occurrence of a value at the output of each gate assuming all the inputs are independent. For example, the probability of occurrence of 1 at the output

of a 2 input AND gate with inputs a , b and output c is: $P(c = 1) = P(a = 1) * P(b = 1)$. These probabilities are estimated from the primary inputs to the output and updated for all the gates in the circuit. Note that the probabilities are measured only once and are computed similar to COP [9] in determining the controllability measures of gates in the circuit.

Let the probability of occurrence of a value at a gate g_i in the cut-set be $P(g_i)$. Assuming that the values at the individual gates occur independent of each other, the probability of occurrence of a particular cut-set (set of specified gates together) is

$$P(\text{Cutset}) = P(g_1) \times P(g_2) \times P(g_3) \times \dots \times P(g_n)$$

After each ATPG decision, we compute the cut-set and estimate the probability of its occurrence. With this new probability as a metric, we can decide if we should store a cut-set or not. We store a cut-set only if it has a high probability of occurrence. Cut-sets with low probability of occurrence are ignored since they are deemed unlikely to be used again. Likewise, while searching for a cut-set in the hash-table, only cut-sets with higher probability of occurrence are considered. A cut-set with low probability of occurrence need not be searched in the hash-table since it would not have been stored in the first place. It may also be noted that the probability of occurrence of a cut-set decreases with an increase in its size, from the above equation. This explains the phenomenon that it is easier to obtain short cut-sets that are equivalent, whereas equivalent long cut-sets cannot be easily formed. Furthermore, the long cut-sets usually occur near the terminal nodes of the decision tree. Even if we ignore these cut-sets, the ATPG engine is likely to make the remaining few decisions to reach the terminal nodes without significant time overhead. On the contrary, cut-sets in the upper portions (near the root) of the decision tree are very important and would cost more if they are missed.

In order to reduce the number of cut-sets, we need to decide a CUTSET_THRESHOLD. All cut-sets with probability of occurrence less than the specified threshold will be ignored. However, deciding the CUTSET_THRESHOLD is non-trivial. In our techniques, we chose the probability of occurrence of the objective as the CUTSET_THRESHOLD. If a cut-set has a higher probability of occurrence than the objective, it is likely to occur often during ATPG. Experimental results show that the number of cut-sets stored is reduced, without much loss in speed of the ATPG engine.

6 Experimental Results

The above techniques were implemented in C++ and integrated into a PODEM based ATPG to generate all the solutions that satisfy a target objective. Success driven learning was integrated into the ATPG engine. In addition to the solution cut-sets suggested in [1], the conflict cut-sets were also stored to learn from the conflict subspaces encountered during search. Experiments were conducted to

compute the one-cycle preimage for some large ISCAS '89 and ITC '99 benchmark circuits. Random conjectures of a set of flip-flops were chosen and set to random 0/1 values. The ATPG engine was used to find all the previous states that can lead to the target state in one transition. A backtrack limit of 1,000,000 was set for the ATPG engine. The experiments were conducted on a 1.8 GHz Pentium 4 machine with 512MB RAM, running the Linux Operating System.

The techniques proposed in this paper aim at enhancing an ATPG based framework for design verification problems and hence we compare our techniques with existing heuristics for an ATPG based framework only. We compare the results of our guidance heuristic with those of conventional heuristics - distance based, COP, and SCOAP measures [8] - in Tables 1 and 2. For each circuit, these three guidance heuristics and two of our heuristics were used individually for computing all solutions that satisfy a target objective. In our first heuristic - *Stat-Conn*, only the static connectivity measures were used. In our second heuristic - *Dyn-Conn*, the dynamic connectivity measures were updated after each decision and the backtrack routine was guided according to these measures. For each guidance heuristic, the number of solution cubes obtained, number of backtracks, number of Free BDD nodes used to store the solution-set, and the time taken for an 'all-solutions ATPG engine' are reported. Only large circuits are considered, since the number of BDD nodes in small circuits is usually small and the corresponding BDDs can be easily stored.

According to the results, it was observed that there is a reduction in the number of final BDD nodes for the proposed technique. In s3384-1, Dist and COP fail to find all the solutions. Although SCOAP manages to find all the solutions, the number of BDD nodes is reduced by an order of magnitude in our connectivity based measures. In s38584-1, where SCOAP *fails* to find all the solutions, the connectivity based measures find all the solutions. In s38584-2, the number of backtracks is less for Dyn-Conn than Stat-Conn. However, longer cut-sets occur during ATPG and the estimation of dynamic-connectivity measures after each decision probably slows down the ATPG engine. But the payoff is in the number of BDD nodes that is reduced to almost half when compared to Stat-Conn. In some cases, such as s15850-2 and s38584-3, when all the other three methods fail, Dyn-Conn is able to compute all the solutions. In Table 2, the results for ITC '99 circuits also confirm that we can obtain a compact solution-set using our guidance measures as compared to Dist, COP and SCOAP. Although the connectivity based measures decide on related variables together, the first set of variables that are targeted are not tuned by these methods. These set of variables may result in a conflict or influence all the other solutions that occur during the ATPG. They are sometimes targeted by other heuristics, where they perform better than connectivity based measures as seen in s15850-1. In s15850-1, although the number of backtracks is less for SCOAP, we obtain a compact solution-set as good as the one obtained using SCOAP measures.

Table 1. All Solutions ATPG for ISCAS '89 circuits

Circuit	Heuristic	#solns	#bkrks	#nodes	T (s)	Circuit	Heuristic	#solns	#bkrks	#nodes	T (s)
s3384 - 1	Dist	464M	Abort			s5378 - 1	Dist	0	361	0	0.28
	COP	19M	Abort				COP	0	59	0	0.26
	SCOAP	1.8G	2.6K	2K	0.69		SCOAP	0	93	0	0.26
	Stat-Conn	1.9G	1K	396	0.43		Stat-Conn	0	8	0	0.25
	Dyn-Conn	1.7G	518	398	0.62		Dyn-Conn	0	3	0	0.26
s3384 - 2	Dist	0	Abort			s5378 - 2	Dist	2M	3K	1.3K	0.71
	COP	0	Abort				COP	82K	1.1K	525	0.5
	SCOAP	0	Abort				SCOAP	218K	1.9K	779	0.58
	Stat-Conn	0	Abort				Stat-Conn	6.3M	887	392	0.49
	Dyn-Conn	108M	158K	2.5K	78.58		Dyn-Conn	299K	389	178	0.66
s3384 - 3	Dist	0	Abort			s5378 - 3	Dist	43K	302	303	0.42
	COP	0	Abort				COP	26K	521	505	0.44
	SCOAP	1G	69K	57K	10.64		SCOAP	32K	467	440	0.43
	Stat-Conn	1.1G	16K	7K	2.19		Stat-Conn	13K	286	278	0.41
	Dyn-Conn	1.8G	9K	1.4K	4.7		Dyn-Conn	19K	178	167	0.52
s13207 - 1	Dist	798K	349	342	0.44	s15850 - 1	Dist	308M	Abort		
	COP	241K	255	250	0.42		COP	2.4G	Abort		
	SCOAP	467K	338	339	0.44		SCOAP	605M	38K	27K	7.25
	Stat-Conn	974K	782	784	0.51		Stat-Conn	3.8G	Abort		
	Dyn-Conn	952K	195	193	0.64		Dyn-Conn	1.2G	313K	26K	59.92
s13207 - 2	Dist	1.5G	Abort			s15850 - 2	Dist	3.5G	Abort		
	COP	2.5G	Abort				COP	3.3G	Abort		
	SCOAP	1.4G	40K	7K	9.24		SCOAP	1.6G	Abort		
	Stat-Conn	2.5G	Abort				Stat-Conn	3.6G	Abort		
	Dyn-Conn	889M	3.8K	2.7K	5.22		Dyn-Conn	1.7G	177K	101K	269.99
s13207 - 3	Dist	0	1526	0	1.32	s15850 - 3	Dist	0	Abort	0	69.83
	COP	0	2220	0	1.45		COP	0	2	0	0.68
	SCOAP	0	1836	0	1.41		SCOAP	0	2	0	0.68
	Stat-Conn	0	100	0	0.41		Stat-Conn	0	1	0	0.71
	Dyn-Conn	0	215	0	1.06		Dyn-Conn	0	1	0	0.71
s9234 - 1	Dist	537M	Abort			s38584 - 1	Dist	104M	20K	20K	13
	COP	41M	43K	31K	5.19		COP	1G	Abort		
	SCOAP	367M	34K	28K	4.32		SCOAP	1.1G	Abort		
	Stat-Conn	3.9G	Abort				Stat-Conn	1.3G	51K	28K	29.71
	Dyn-Conn	1.6G	8K	6.5K	7.81		Dyn-Conn	1.4G	21K	8.5K	27.57
s9234 - 2	Dist	0	1.9K	0	0.56	s38584 - 2	Dist	1.4G	4.5K	4K	1.8
	COP	0	409	0	0.36		COP	1G	Abort		
	SCOAP	0	1.6K	0	0.53		SCOAP	1.5G	3.8K	3.2K	1.81
	Stat-Conn	0	15	0	0.32		Stat-Conn	1.2G	3.2K	3.1K	1.67
	Dyn-Conn	0	19	0	0.34		Dyn-Conn	1.5G	1.7K	1.7K	7.79
s9234 - 3	Dist	0	42K	0	5.8	s38584 - 3	Dist	3.5G	Abort		
	COP	0	Abort				COP	1.3G	Abort		
	SCOAP	0	Abort				SCOAP	3.7G	Abort		
	Stat-Conn	0	24K	0	3.02		Stat-Conn	1.2G	Abort		
	Dyn-Conn	0	7K	0	6.28		Dyn-Conn	1.9G	46K	45K	203.05

Note a) #solns: number of solution cubes

Note b) Different guidance heuristics lead to different solution cubes

We conducted a second set of experiments to verify the effectiveness of the *cut-set metric* for reducing the cut-set storage. The ATPG environment was set up similar to the previous experiments to generate all solutions. In addition, the number of cut-sets we can store was limited to 300,000. After each decision, the cut-set was determined and its probability of occurrence was estimated. Cut-sets with probabilities less than a fixed CUTSET_THRESHOLD were ignored. *The probability of occurrence of the objective was fixed as the CUTSET_THRESHOLD.* The corresponding results are tabulated in Table 3. For each circuit shown in column 1, columns 2-6 report the number of solutions, number of BDD nodes, number of stored cut-sets, number of times the stored cut-sets are used and the time taken without using the cut-set metric. The number of times the stored-cutsets, as a whole, are used is counted every time a hash-hit for a cut-set is suc-

cessful. Columns 7-11 report the results for the same ATPG by storing only the cut-sets with a high probability of occurrence.

For the objectives in smaller circuits like s349, s444 and s526, the cut-sets that are stored is reduced. However, since they are very small in number, all the cut-sets can be easily stored. On the other hand, in cases where large number of cut-sets are encountered, as in s1423.1, s4863, b04 and b11, the number of cut-sets is significantly reduced without significant loss in time. The main advantage in using the cut-set metric is exhibited in the cases of s1423 and s9234. Without using the proposed cut-set metric, all the cut-sets are exhausted and the ATPG engine fails to complete. The number of times the stored-cutsets are used varies according to the target objective in the two methods. Sometimes, longer cut-sets in the lower part of the decision tree are NOT

Table 2. All Solutions ATPG for ITC '99 circuits

Circuit	Heuristic	#solns	#bktrks	#nodes	T (s)	Circuit	Heuristic	#solns	#bktrks	#nodes	T (s)
b04	Dist	20K	3K	1.2K	0.49	b10	Dist	118	107	85	0.24
	COP	32K	3.5K	2.2K	0.5		COP	33	60	45	0.23
	SCOAP	5M	Abort				SCOAP	36	59	50	0.24
	Stat-Conn	26K	1.5K	617	0.34		Stat-Conn	22	39	33	0.23
	Dyn-Conn	27K	2K	906	0.71		Dyn-Conn	31	46	37	0.23
b12 - 1	Dist	936K	23K	10K	1.99	b13 - 1	Dist	594	46	44	0.24
	COP	8.5K	1.8K	1.5K	0.36		COP	432	175	149	0.24
	SCOAP	445K	38K	7.5K	3.3		SCOAP	2K	225	207	0.24
	Stat-Conn	128K	978	793	0.31		Stat-Conn	1.2K	113	93	0.24
	Dyn-Conn	478K	504	295	0.39		Dyn-Conn	612	70	45	0.25
b12 - 2	Dist	261K	Abort			b13 - 2	Dist	36K	2K	1K	0.32
	COP	1.7K	6K	5K	0.65		COP	540	241	187	0.25
	SCOAP	1M	66K	9.6K	6		SCOAP	9.5K	2.6K	1.6K	0.35
	Stat-Conn	51K	1.7K	740	0.38		Stat-Conn	3.5K	128	103	0.23
	Dyn-Conn	114K	764	338	0.44		Dyn-Conn	2K	124	121	0.25
b12 - 3	Dist	38K	3.3K	2.3K	0.52	b13 - 3	Dist	112	177	106	0.34
	COP	8.8K	1.3K	1K	0.36		COP	21	91	78	0.25
	SCOAP	16K	915	734	0.32		SCOAP	72	133	96	0.24
	Stat-Conn	30K	1.3K	795	0.35		Stat-Conn	60	118	55	0.25
	Dyn-Conn	131K	717	486	0.43		Dyn-Conn	21	63	42	0.25

Note a) #solns: number of solution cubes

Note b) Different guidance heuristics lead to different solution cubes

stored in the second method. However, the ATPG quickly makes the fewer decisions to reach the terminal nodes. This results in fewer hash-hits in the second method without significant loss in time as seen in b11. On the other hand, some cut-sets that are used fewer times in the first method are not available to the second method. So the available cut-sets are used many times in the second method, leading to an increase in the number of times the stored-cutsets are used, as seen in s1269 and s15850.1.

Although the metric is helpful to obtain a probabilistic estimate for the use of a cut-set, a low value does not completely guarantee that the cut-set will not be re-used again. This uncertainty may lead to losing useful cut-sets occasionally. In such cases, the ATPG engine may have to re-search a previously visited search space and take more time as seen in s15850.1. This phenomenon may increase as the size of the circuit increases due to correlation of gates in the circuit.

7 Conclusion and Future Work

In this paper, we explored the factors that govern an all-solutions ATPG engine. While conventional guidance heuristics for ATPG are suitable for finding a single solution quickly, they do not account for the complete-solution set. We presented a new *connectivity based guidance heuristic* for an 'all-solutions ATPG engine' that reduces the size of the Free BDD to store all solutions. We also developed a probabilistic *cut-set metric* to determine the future use of stored information, learnt during success driven learning. Based on the cut-set metric, we showed that the knowledge base can be reduced depending on the CUT-SET-THRESHOLD determined for the cut-set. Experi-

mental results show that we can achieve significant reductions in memory and our techniques can guide the ATPG engine to find all the solutions where conventional guidance heuristics fail. In the future, we plan to apply partitioning of variables to obtain a better variable order. We also plan to consider the effect of signal correlations to improve the guidance for ATPG and develop a dynamic cut-set threshold value to get a better approximation for the cut-set usability. With these feasible techniques, we envision a robust ATPG engine that can be effectively used for model checking sequential circuits as well as for improving the defect coverage of a test-suite.

References

- [1] S. Sheng and M. S. Hsiao, "Efficient Pre-Image Computation Using a Novel Success-Driven ATPG", *Proc. DATE* 2003, pp. 822-827.
- [2] P. A. Abdulla, P. Bjesse and N. Een, "Symbolic Reachability Analysis based on SAT-Solvers", *Proc. TACAS*, 2000, pp. 411-425.
- [3] B. Li, M. S. Hsiao, S. Sheng "A novel SAT all-solutions solver for efficient preimage computation," in *Proc. DATE*, 2004, pp. 272-277.
- [4] K. L. McMillan, "Applying SAT methods in unbounded symbolic model checking", *Proc. CAV*, 2002, pp. 250-264.
- [5] H.J. Kang, I.C. Park, "SAT-Based Unbounded Symbolic Model Checking", *Proc. DAC*, 2003, pp. 840-843.
- [6] M.K. Iyer, G. Parthasarathy, L. -C. Wang and K. T. -Cheng, "On the Development of ATPG based SAT Checker", *Proc. MTV*, 2002

Table 3. Efficiency of Our Cut-set metric

ckt	Without Cut-set Metric					With Cut-set Metric				
	#solns	#nodes	#cut-sets	#times-used	time(s)	#solns	#nodes	#cut-sets	#times-used	time(s)
s349	200	193	366	207	0.26	200	193	101	125	0.27
s444	24	65	81	19	0.25	24	65	54	18	0.24
s526	54	84	94	41	0.25	54	84	43	35	0.24
s4863	0	0	132K	8K	70.71	0	0	125K	8K	71.46
s1488	57	95	172	59	0.26	57	95	3	38	0.28
s1494	9	29	95	36	0.26	9	29	7	45	0.28
s1269	183K	113K	236K	120K	39.89	183K	113K	148K	204K	57.61
s1423	All	cut-sets	exhausted			330K	2.5K	150K	50K	64.73
s1423.1	912	810	123K	954	23.66	912	810	110K	825	23.10
s5378	805M	272	462	211	0.56	805M	272	444	211	0.56
s9234	All	cut-sets	exhausted			0	0	200K	190K	491.71
s9234.1	0	0	84K	56K	133.19	0	0	77K	59K	151.24
s15850.1	981M	3K	4.8K	2.7K	14.1	981M	3K	2.2K	8K	38.97
s15850	0	0	12K	10K	41.44	0	0	10K	11K	48.66
b04	6.5K	6.8K	102K	2.6K	254.2	6.5K	6.8K	20K	2.3K	282.49
b11	559	1.5K	15K	1.3K	35.72	559	1.5K	189	705	36.32
b12	65K	1.2K	1.2K	576	6.18	65K	1.2K	1K	576	6.18

Note a) #solns: number of solution cubes

Note b) Dyn-Con is used. Hence, the same #soln cubes are obtained for both techniques

- [7] F. Lu, L.C. Wang, K.T.Cheng, R.C Huang, "A Signal Correlation Guided ATPG Solver And Its Applications For Solving Difficult Industrial Cases", *Proc. DAC*, 2003, pp. 436-441.
- [8] M. Abramovici, M. A. Breuer and A. D. Friedman, "Digital Systems Testing and Testable Design", *IEEE Press*, NJ, 1990.
- [9] F. Brglez, "On Testability of Combinational Networks", *Proc. Intl. Symp. Circuits and Systems*, 1984, pp. 221-225.
- [10] L. H. Goldstein, "Controllability/Observability Analysis of digital circuits", *IEEE Trans. Circuits and Systems*, Vol. 26, Sept 1979, pp. 685-693.
- [11] S.C. Seth, L. Pan and V. D. Agrawal, "PREDICT: Probabilistic estimation of digital circuit Testability", *Proc. FTCS*, 1985, 220-225.
- [12] S.C. Chang, W.B. Jone, S.S. Chang, "TAIR:Testability Analysis by Implication Reasoning", *IEEE Trans. on Computer Aided Design*, Vol. 19, No. 1, Jan 2000, pp. 152-160.
- [13] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability", *IEEE Trans. Computers*, Vol. 48, No. 5, 1999, pp. 506-521.
- [14] L. Zhang, C. F. Madigan, M. H. Moskewicz and S. Malik, "Efficient Conflict Driven Learning in a Boolean Satisfiability Solver", *Proc. ICCAD*, 2001, pp. 279-285.
- [15] M.K. Iyer, G. Parthasarathy, K.-T.Cheng, "SATORI - A Fast Sequential SAT Engine for Circuits", *Proc. ICCAD*, 2003.
- [16] R. Rudell, "Dynamic Variable Ordering for ordered binary decision diagrams", *Proc. ICCAD*, 1993, pp. 42-47.
- [17] M. Thornton, J. Williams, R. Drechsler, and N. Drechsler, "Variable re-ordering for shared binary decision diagrams using output probabilities", *Proc. DATE*, 1999, pp. 758-759.
- [18] S. Panda and F. Somenzi, "Who are the variables in your neighbourhood", *Proc. ICCAD*, 1995, pp. 74-77.
- [19] P.Y. Chung, I. N. Hajj and J.H. Patel, "Efficient Variable Ordering Heuristics for Shared ROBDD", *Proc. ISCAS*, 1993, pp. 1690-1693.
- [20] F. Aloul, I. Markov and K. Sakallah "MINCE: A Static Global Variable-Ordering for SAT and BDD", *IWLS*, 2001, pp. 281-286.
- [21] F. Aloul, I. Markov and K. Sakallah "FORCE: A Fast and Easy-To-Implement Variable Ordering Heuristic", *GLSVLSI*, 2003, pp. 116-119
- [22] J. Gregov, C. Meinel, "Efficient Boolean Manipulation with OBDD's can be Extended to FBDDs", *IEEE Trans. on Computers*, Vol. 43 No. 10, Oct 1994, pp. 1197-1209.
- [23] J. Bern, C. Meinel, A. Slobodova, "Some heuristics for Generating Tree-like FBDD Types", *IEEE Trans. on Computer Aided Design*, Vol. 16, No. 1, Jan. 1996, pp. 127-130.
- [24] M. H. Schulz, E. Trischler and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System", *IEEE Trans. CAD*, Vol.7, No.1, 1988, pp. 126-137.
- [25] J. Giraldi and M. L. Bushnell, "EST: The New Frontier in Automatic Test-Pattern Gen.", *Proc. DAC*, 1990, pp. 667-672.
- [26] W. Kunz and D. Pradhan, "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD-problems", *IEEE Trans. on CAD*, Vol 13 (9), 1994, pp. 1143-1158.
- [27] J. P. Marques-Silva and K. A. Sakallah, "Dynamic Search-Space Pruning Techniques in Path Sensitization", *Proc. DAC*, 1994, pp. 705-711.
- [28] I. Hamzaoglu and J. H. Patel, "New Techniques for Deterministic Test Pattern Gen.", *Proc. VTS*, 1998, pp. 446-452.
- [29] C. Wang, S.M. Reddy, I. Pomeranz, L. Xiang and J. Rajski, "Conflict driven techniques for improving deterministic test pattern generation", *Proc. ICCAD*, 2002, pp. 87-93.
- [30] K. Chandrasekar and M.S. Hsiao, "ATPG based preimage computation: Efficient search-spacing pruning using ZBDD", *Proc. of HLDVT*, 2003, pp. 117-122.
- [31] I. Pomeranz and S.M. Reddy, "An Efficient Nonenumerative Method to Estimate Path Delay Fault Coverage", *ICCAD*, 1992, pp. 560-567.