# Untestable Fault Identification through Enhanced Necessary Value Assignments *

Vishnu C. Vimjam
ECE Dept., Virginia Tech
Blacksburg, VA - 24061
vvimjam@vt.edu

Manan Syal
ECE Dept., Virginia Tech
Blacksburg, VA - 24061
msyal@vt.edu

Michael S. Hsiao
ECE Dept., Virginia Tech
Blacksburg, VA - 24061
mhsiao@vt.edu

## ABSTRACT

In this paper, we propose novel low-cost methods that combine *static* logic implications and binary resolution to significantly increase the number of non-trivial signal relations learned from the circuit. The proposed method first applies resolution techniques to learn new static single-node implications and then uses them to learn powerful multi-node implications. All the newly learned relations help in extracting more necessary assignments for a given fault, potentially increasing the chance for a conflict to occur among the necessary assignments. Experimental results on ISCAS89 and ITC99 benchmarks show that our method can identify significantly more untestable faults compared to existing *non branch-and-bound* based techniques.

## Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing, Fault Tolerance

## General Terms

Algorithms, Design

## Keywords

Untestable faults, Implications

## 1. INTRODUCTION

Untestable single stuck-at-faults often present tremendous challenge for ATPG engines because of the non-existence of a single input vector/sequence that can detect these faults. This problem becomes further complicated for sequential circuits because of the additional process required in justifying a state that enables the detection of a fault. Identifying the untestable faults through low-cost techniques can be extremely beneficial to avoid the huge processing times incurred by the ATPG engines upon handling these hard untestable faults.

* supported in part by NSF Grants 0196470, 0098304 and 0305881


*GLSVLSI'05,* April 17–19, 2005, Chicago, Illinois, USA.


The Single Fault Theorem presented in [1] unrolls the sequential circuit for a given number of time-frames and verifies through combinational ATPG whether a fault is detectable by injecting it in the right-most time-frame. If no vector can detect the fault in such a setup, then the fault is guaranteed to be sequentially untestable. On the other hand, if the fault-effect can be propagated to a primary output or flip-flop in the right-most time-frame, nothing can be concluded. Usually, a larger number of unrolled time-frames leads to more untestable faults found. Three new procedures extending the single-fault theorem were introduced in [2] to aid the identification of a larger set of untestable faults. While these techniques reduce the sequential ATPG problem to that of a combinational ATPG, the complexity is still exponential in the circuit size in the worst case.

A fault-independent algorithm, FIRE, is proposed in [3] for combinational circuits that identifies redundant faults requiring conflicting values in the circuit. Unlike branch-and-bound methods that have exponential complexity, FIRE's complexity is polynomial in the circuit size. In its implementation, redundant faults that require conflicts on a single line were reported. Since no branch-and-bound search is involved, it has been effective for a number of circuits. This concept has been extended to identify sequentially untestable faults in [4] by formulating new validation rules for propagating uncontrollability and unobservability conditions across frame boundaries.

Illegal states are identified using BDDs in [5] which are then used to identify sequentially untestable faults. The performance of this technique depends on the efficiency of capturing as many illegal states as possible and hence is more attractive for smaller circuits. Impossible value combinations between a gate output and its fanins have been used in [6] to identify faults that require an locally impossible value combination as untestable. This technique has been extended in [7] to obtain multi-line conflicts between a gate and its immediate justification frontier which can capture more untestable faults.

MUST, proposed in [8], aims at finding conflicts among necessary multiple stem assignments for a given fault. Since it targets each fault at a time, the new necessary assignments derived in the process can be effective in leading to a conflict if a fault is untestable. It has been shown in [8] that the faults identified as untestable by MUST are potentially hard for sequential ATPG engines. Though powerful for small and medium-sized circuits, the memory requirement for storing all the necessary values for the faults limit the algorithm's effectiveness for large circuits.

It can be seen that the performance of most of the untestable fault identification algorithms depend on the underlying set of implications deduced from the circuit. More non-trivial relations available allows for more untestable faults (if they exist) that an algorithm can identify. Techniques such as Recursive learning [9] and Ex-

tended Backward learning [11] have been proposed in the past for computing more implications. Two procedures employing recursive learning have been developed in [10] to check the inconsistency of the set of necessary assignments for a fault. While learning of unlimited recursion depth can capture all the necessary assignments for a given line, it often becomes prohibitive due to the exponential time complexity in the recursion depth. Hence a smaller recursion depth is often desirable. Finally, finding (untestable) faults that become untestable for all the four logic combinations of a pair of gates was attempted in [13]. To avoid exploring all gate pairs, the gates that lie locally close (e.g., in the same circuit level) were targeted.

In this paper, we propose two novel, low-cost techniques that can increase the set of necessary assignments for a given fault. We combine binary resolution and static learning to deduce powerful implications whose computation can be integrated with the computation of either extended backward or recursive learning. Using the learned implications, our technique aims at quickly extracting non-trivial multi-node relations that can be extremely useful for deducing more implications dynamically for a given fault. In addition, we provide heuristics to efficiently store the necessary assignments for each fault. Application of our technique to ISCAS and ITC benchmarks show that significantly more untestable faults can be identified.

The rest of the paper is organized as follows: Section 2 describes the preliminaries; Section 3 presents our contribution and algorithms. We provide the overall algorithm in Section 4 and the experimental results are reported in Section 5. In Section 6, we conclude the paper.

# 2. PRELIMINARIES

## 2.1 Terms and Notations

Throughout the paper, we use the following notations: A Boolean gate is denoted with an upper-case alphabet such as $X, X1$ *etc.* and a Boolean logic value 0 or 1 is denoted with the lower-case alphabet $v$. A node is a value assignment to a gate (0 or 1). If a circuit has $n$ gates, then it has a total of $2n$ nodes. In general, we represent any node as $X_v$ where $v \in \{0, 1\}$. Specifically, $X_0(X_1)$ represents the node for gate $X$ with value $= 0(1)$. We use the symbol & to refer to the union of the nodes.

A single-node implication refers to the relationship between a pair of nodes, i.e., one node implies another node. Multi-node relations refer to a set of nodes together implying a single node. For an AND gate $C$ with two inputs $A$ and $B$, $A_0 \rightarrow C_0$ and $B_0 \rightarrow C_0$ are single-node implications, where as $(A_1 \& B_1) \rightarrow C_1$ is a multi-node relation. We use a curly-braced set to denote a conflicting scenario among the set of nodes. For example, the set $\{X_{v1}, Y_{v2}\}$ denotes that gate assignments $X = v1$ and $Y = v2$ cannot exist *simultaneously* inside the circuit. Note that a conflicting scenario involving $n$-nodes has $n$ multi-node relations within it. In general, we use both the terms *conflicting scenario* and *multi-node relation* inter-changeably.

A gate is said to be specified if it is assigned a logic value $v$. If its value is unknown, then the gate is said to be unspecified. A specified gate is said to be unjustified (by its inputs) if the current assignments (if any) of its inputs do not justify the output value of the gate. For example, an OR gate assigned to value 1 with none of its inputs assigned to value 1 is an unjustified gate.

## 2.2 Implication Graph

An implication graph $G(V, E)$ of a circuit consists of vertices, $V$, which belong to the set of nodes in the circuit, and directed edges, $E$, that represent single-node implications. If an assignment $X = 0$ (say) implies $Y = 1$, a directed edge from node $X_0$ to node $Y_1$ and the contra-positive edge ($Y_0$ to $X_1$) are added to the graph. Adding a new implication edge to the graph can have quadratic impact on the total number of implications due to their transitive property. Using a edge-weighted graph structure to store the implications helps in (i) obtaining implications through several time-frames without actually unrolling the circuit and (ii) provides a compact way to store all the learned implications. For more details on the construction of an implication graph, the reader is referred to [12]. Below, we review the existing implication learning techniques.

**Extended Backward Learning (EBL):** For a node $N_v$, let a specified gate $G$ be unjustified with $n$ unspecified inputs $X1, X2...Xn$. Let $v_1, v_2...v_n$ be the assignments to each unspecified input respectively such that any specified input $Xi$ can justify gate $G$. Now, the intersection of gate assignments obtained by logic simulating the sets $(N_v \& X1_{v1})$, $(N_v \& X2_{v2})$,..., $(N_v \& Xn_{vn})$ are also implications of node $N_v$. For the circuit shown in Figure 1, we initially have $X_1 \rightarrow A_1$ which makes gate $A$ unjustified. Performing extended backward learning on its two unspecified inputs helps us to learn that $X_1 \rightarrow Y_1$. Via the contra-positive law, the implication $Y_0 \rightarrow X_0$ is also learned. Often these contra-positive implications become very helpful in increasing the total number of non-trivial implications in the circuit.

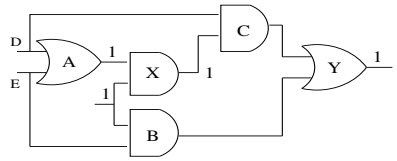


**Figure 1: Illustration of Implication Learning**

**Recursive Learning (RL):** Recursive learning generalizes the concept of an unjustified gate to make *precise* forward implications and aims at capturing the implications common in all the justification scenarios. Higher recursion depths help in extracting more complicated implications from the circuit but are prone to an explosion in time.

At this point, we note that implications obtained through EBL are strictly a subset of those obtained through RL because of the generalized *unjustified gate* definition used in [9]. Nevertheless, our current implementation is based on the unjustified gate concept of EBL and can also be applied along with RL.

## 2.3 Review of Recurrence Relations [7]

Recurrence relations are used to identify gates that are sequentially unachievable to a certain value (0 or 1). If a node $X_v$ in current time-frame (say, $t$) implies the same node $X_v$ in the previous time-frame (say, $t-1$), then random simulation is performed to check whether the opposite node *i.e.,* $X_{\bar{v}}$ is ever achievable. If $X = \bar{v}$ is possible, then it can be concluded that the node $X_v$ is sequentially unachievable and all the faults that require $X = v$ for their detection are sequentially untestable. For more details on recurrence relations, the reader is referred to [7].

## 2.4 Review of MUST [8]

The MUST procedure for untestable fault identification involves two phases. The first is a preprocessing phase during which the subset of necessary assignments is computed for each fault. Es-

sentially, if a fault $f$ becomes untestable due to a stem assignment $X = v$, then $X = \bar{v}$ is stored as a necessary stem assignment for $f$. In the second phase, all such required stem assignments for each fault are injected onto the circuit and simulated. If an implication conflict occurs or if the fault cannot be activated or observed anymore, then it is said to be untestable; else nothing can be concluded.

# 3. RESOLUTION-BASED LEARNING

Binary resolution is a deductive step employed to obtain powerful relationships by quantifying a variable. Given two set(s) of conflicting scenarios with one set involving a node $G_v$ and the other involving the node $G_{\bar{v}}$, gate $G$ can be eliminated from the scenarios to obtain a new conflicting relation. For example, if $\{X_v, Y_v, G_v\}$ and $\{Z_v, G_{\bar{v}}\}$ are two conflicting scenarios, then the new relation $\{X_v, Y_v, Z_v\}$ can be obtained by taking the union of the two sets and eliminating the two opposing nodes of $G$.

In section 3.1, we use this concept to deduce powerful single-node implications. In section 3.2, a series of resolutions are then applied using the conflicting scenarios in the circuit to learn non-trivial multi-node relations, which cannot be derived directly from the single-node implications.

## 3.1 Learning New Single-Node Implications

Consider a gate $X$ in the circuit. Let $S$ represent the set of implications in the intersection of two opposing nodes related to gate $X$, *i.e.*, $X_0$ and $X_1$. Consider $S$ to be non-empty, and let $G_v$ be a node in $S$. Since $G_v$ is implied by both $X_0$ and $X_1$, $G_{\bar{v}}$ should be an impossible node, *i.e.,* $G$ is a constant gate achievable to only one value $v$. However, if the circuit has no constant gates, then no such sets would exist. Our aim is to find conditional constants, whereby under the presence of some other node $Y_v$, the set $S$ is no longer empty for a target gate $X$. In other words, we want to find if the multi-node relations $(X_0 \& Y_v) \rightarrow Z_v$ and $(X_1 \& Y_v) \rightarrow Z_v$ hold. In such a case, it can be concluded that $Y_v \rightarrow Z_v$ according to the following Lemma.

**Lemma 1:** If the intersection of two mutually exclusive nodes $X_0$ and $X_1$ is non-empty under a condition $Y_v$, then the nodes in the intersection are implicants of node $Y_v$. In other words, if $(X_0 \& Y_v) \rightarrow \{S\}$ and $(X_1 \& Y_v) \rightarrow \{S\}$, then $Y_v \rightarrow \{S\}$, where $\{S\}$ is the set of common implications.

**Proof:** The proof is straight forward with the application of binary resolution. Resolving on gate $X$ eliminates the nodes $X_0$ and $X_1$ from the above multi-node relations, thus resulting in $Y_v \rightarrow \{S\}$. □

We call node $Y_v$ as the *base node*, gate $X$ as the *resolver* and nodes $X_0$, $X_1$ as the *resolving nodes*. Given Lemma 1, one can aim at finding base nodes such that the intersecting set $S$ of the resolving nodes is non-empty. But, in order to avoid selecting too many base nodes for a given resolver gate, we formulate this problem in the reverse way to find good resolver gates for a given base node. Recall that during the application of EBL for a node $Y_v$, we find unjustified gates and use their unspecified inputs for justifying them. We find that such unspecified inputs for an unjustified gate are good *candidate resolver* gates for the base node $Y_v$. This heuristic selection helps in (1) capturing efficient reconvergences of the resolving nodes through the base node and (2) keeping the simulation overhead as small as possible since half of the work is already being done during the EBL computation. To further reduce the simulation costs, the two resolving nodes can be bit-packed and simulated together.

We demonstrate this type of resolution with the help of an example shown in Figure 2. Let the implications $E_1 \rightarrow C_0$, $E_1 \rightarrow H_1$, $G_1 \rightarrow I_1$, $K_0 \rightarrow J_1$ and $D_0 \rightarrow F_0$ already exist in the implication graph. Consider the base node $X_0$ with direct implications $A_0$, $B_0$ and $K_0$. None of them can imply any other gates even when extended backward learning is iterated. Consider gate $E$ which is one of the unspecified inputs of $A$. Figure 2 shows how the node combination ($X_0$ & $E_1$) leads to a conflict around gate $B$. Thus we can conclude that $X_0 \rightarrow E_0$ and hence $E_1 \rightarrow X_1$ via the contrapositive law.

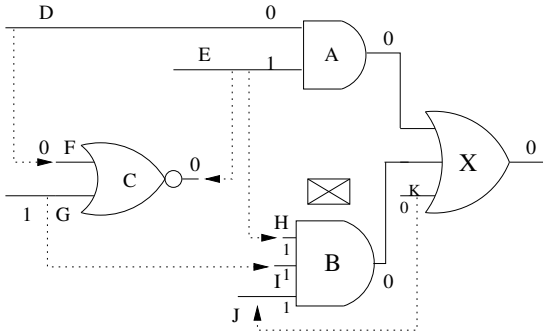


**Figure 2: Implications through Justification Enumeration**

While the above implication can be learned with RL with a deep level of recursion, our technique can find it very quickly without any recursion. This example emphasizes that our technique can quickly capture non-trivial relations which occur due to the reconvergences inside the circuit. Such internal relations are very powerful, and adding them to the implication graph enables us to capture even more powerful implications during the learning process. Note that such resolution can be performed dynamically for each fault as is done in [15] but is subject to two major drawbacks: first, the implications of the resolving nodes are not established beforehand and hence it misses several crucial relations, and second, checking for common implications at every decision level would need excessive ATPG run times. In the case of static learning such as our method, in the worst case, this is performed quadratic to the number of nodes in the circuit.

So far we have described the case where only a single gate is used for resolution under a base node. Note that in Figure 2, when gate $A$ becomes unjustified for node $X_0$, multi-gate resolution can also be performed using the three sets $(D_0 \& E_0)$, $(D_0 \& E_1)$ and $(D_1 \& E_0)$ because it automatically covers the binary resolutions on the unspecified inputs as well as the complete justification scenarios for $A = 0$. Even though such an enumeration would be intuitively more powerful, it would be expensive for taking the transitive closures on the implication graph as well as for performing logic simulation. For an unjustified gate with $n$ unspecified inputs, $n(2^n - 1)$ transitive closures for the nodes would need to be performed and $2^n - 1$ simulations (corresponding to the $2^n - 1$ justifications) to check if the intersection has any nodes in common. However, for the case when $n = 2$, only six transitive closures and three simulations are needed. As done for Lemma 1, all such 3 justification scenarios can be bit-packed and simulated together to enable faster simulation. Hence, we resort to such an enumeration for $n = 2$ but avoid it for $n > 2$ to keep the processing times low.

Next, we describe our resolution techniques for obtaining efficient multi-node implications using the learned single-node implications.

## 3.2 Learning Multi-Node Implications

We refer to the multi-node relation between a gate and its immediate inputs as a Primary multi-node relation. For example, $(A_1 \& B_1) \rightarrow C_0$ forms the primary multi-node relation for the NAND gate $C$ with two inputs $A$ and $B$. The number of single-node implications in a circuit with $n$ gates is limited to $2n(n-1)$, which is only quadratic in terms of the circuit size. On the other hand, multi-node relations for a given circuit can be exponential in the number of gates, since they can involve all possible combinations of all gates. For example, the number of multi-node relations around an OR gate $Y$ with $n$ inputs $X1$, $X2$, ..., $Xn$ can be as large as $I(X1_1) \times I(X2_1) \times ... \times I(Xn_1) \times I(Y_0)$, where $I(X_v)$ represents the number of single-node implications of the node $X_v$. However, since the primary multi-node relation for the OR gate is sufficient to deduce all such multi-node relations, they do not help in increasing the necessary assignments for any fault. Instead, one would be interested in extracting non-trivial multi-node relations that are not covered by the primary relations.

Our algorithm targets learning of such relations. Note that multi-node relations are, in general, less powerful compared to single-node implications and hence low-cost algorithms should be employed to identify them. The method we propose is based on simple traversals on the implication graph and does not involve computation intensive tasks such as logic simulation. We describe our learning in two-phases as given below.

### *3.2.1 Phase I: Obtaining Conflicting Scenarios:*

In this phase, we extract conflicting scenarios for each node in the circuit. Figure 3 shows four such scenarios for AND gates, and the discussion below can be extended to all other gate types.

Consider Figures 3(a) and 3(b). Let $X_v \rightarrow A_1$ and $Y_v \rightarrow F_0$. Then, $\{X_v, B_1, C_0\}$ and $\{Y_v, D_1, E_1\}$ form two conflicting scenarios. As mentioned before, they are not useful since the primary relations around gates $C$ and $F$ already cover them. Now consider Figures 3(c) and 3(d). Let $W_v \rightarrow G_1$, $W_v \rightarrow H_1$, $Z_v \rightarrow M_1$ and $Z_v \rightarrow N_0$. Then, $\{W_v, I_1, J_0\}$ and $\{Z_v, K_1, L_1\}$ form two other conflicting scenarios. Note that these are obtained due to the resolution on the two implied signals around the gates $J$ and $N$ and hence are more powerful compared to the earlier two scenarios. They can aid in extracting more useful information. For example, if nodes $J_0$ and $I_1$ are required assignments for a fault, then we can deduce that $W_{\bar{v}}$ is also needed.

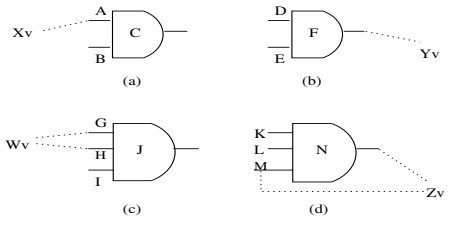


**Figure 3:** **Conflicting Scenarios around gates**

Thus, our phase I can be concluded as finding conflicting scenarios such as those in Figures 3(c) and 3(d) which are obtained as a result of at least two binary resolutions. Data from Phase I can be extracted using just *one* traversal of the implication graph.

### *3.2.2 Phase II: Applying Resolutions in Series:*

In this phase, we apply a series of binary resolutions to scenarios identified in Phase I to obtain non-trivial relations that can be extremely useful for enhancing the set of necessary assignments. We explain our algorithm using the examples illustrated in Figure 4 where the dashed lines represent structural paths in the circuit.

Consider Figure 4(a). Let the implications $A_0 \rightarrow H_0$, $A_0 \rightarrow I_0$, $B_0 \rightarrow D_0$, $B_0 \rightarrow E_0$, $Z_v \rightarrow G_1$, $Z_v \rightarrow K_1$, $X_v \rightarrow C_0$, $Y_v \rightarrow F_0$ and $Y_v \rightarrow J_0$ be present in the circuit. The conflicting scenarios obtained in Phase I for node $A_0$ would be $\{A_0, J_0, K_1\}$ $^{(1)}$ and for node $B_0$ would be $\{B_0, F_0, G_1\}$ $^{(2)}$.

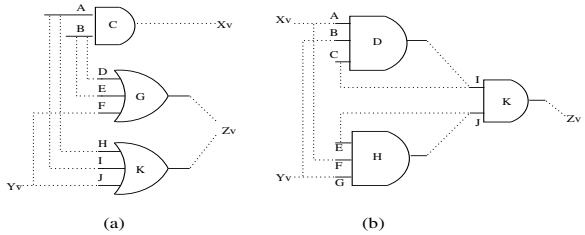


**Figure 4:** **To Illustrate Multi-node Extraction**

Now, consider node $X_v$. Since we are given that $X_v \rightarrow C_0$, either $A_0$ or $B_0$ must be true whenever $X_v$ is true. This leads to the conflicting scenario $\{X_v, A_1, B_1\}$ $^{(3)}$. Eliminating the nodes $A_0$, $A_1$, $B_0$ and $B_1$ via resolution from (1), (2) and (3) would lead to the new scenario $\{X_v, J_0, K_1, F_0, G_1\}$ $^{(4)}$ which is stored in our *first* traversal of the implication graph.

Next, consider node $Y_v$. When $Y_v$ is true, we are told that $F_0$ and $J_0$ are also true, yielding the scenarios $\{Y_v, F_1\}^{(5)}$ and $\{Y_v, J_1\}^{(6)}$. Now eliminating the nodes $F_0$, $F_1$, $J_0$, $J_1$ via resolution from (4), (5) and (6) would lead to $\{X_v, Y_v, K_1, G_1\}^{(7)}$. Note that only $K_1$ and $G_1$ are the two remaining nodes from the Phase I (original) scenarios (1) and (2). This new relation (7) provides an OR situation as follows: Under the condition that $Y_v$ is true, at least one of $G_0$ or $K_0$ has to be true if $X_v$ is true. Such relations are obtained and stored during our *second* traversal of the implication graph.

Finally, consider node $Z_v$. When $Z_v$ is true, we are given that $G_1$ and $K_1$ are true, leading to $\{Z_v, G_0\}^{(8)}$ and $\{Z_v, K_0\}^{(9)}$. Hence $Z_{\bar{v}}$ represent the common implication of $G_0$ and $K_0$. By eliminating the nodes $G_0$, $G_1$, $K_0$ and $K_1$ via resolution from (7), (8) and (9), we can conclude that $\{Y_v, X_v, Z_v\}^{(10)}$ holds true. This new scenario does not involve any of the nodes in the original conflicting scenarios of (1) and (2) and hence we can stop our resolution process.

Note that, in Phase II, the deduced relations (4) and (7) are not useful whereas relation (10) is. Given $Y_v$ and $X_v$ are true in Figure 4(a), a regular ATPG engine cannot deduce that $Z_{\bar{v}}$ is also true without performing additional analysis *via* conventional branch-and-bound process. Similarly, the implied node cannot be deduced in the other two situations, *i.e.,* when ($Y_v$ is true, $Z_v$ is true) and when ($X_v$ is true, $Z_v$ is true). Hence, extracting such non-trivial multi-node relations would be very helpful in deducing new necessary assignments for a fault with very low overhead. At the same time, if a fault *f* requires all the three nodes $Y_v$, $X_v$ and $Z_v$ to be true for its detection, we can immediately conclude that *f* is untestable.

Figure 4(b) illustrates another example, given that the implications $X_v \rightarrow A_1$, $X_v \rightarrow F_1$, $Y_v \rightarrow G_1$, $Y_v \rightarrow B_1$, $I_0 \rightarrow C_1$, $I_0 \rightarrow D_0$, $J_0 \rightarrow E_1$, $J_0 \rightarrow H_0$ and $Z_v \rightarrow K_0$ hold, our technique would learn that $\{Y_v, X_v, Z_v\}$ holds true globally. In both of these examples, we only needed 3 traversals on the implication graph to obtain the relations that involve 3 nodes. Our phase II of the algorithm can thus be generalized as applying a series of resolutions on the phase I scenarios using the single-node implications from the implication graph. If all the nodes present in the (original) conflict-

ing scenarios found in Phase I are eliminated, the resulting relation would be our newly learned multi-node relation. Thus, to extract a relation involving $n$-nodes (if any exists), we would only require $n$ traversals on the implication graph. Note that all such newly obtained relations can be further used iteratively to deduce more powerful multi-node relations. In our current implementation, we perform only one iteration and limit our number of traversals to 3.

# 4. OVERALL ALGORITHM

We have integrated both of the proposed learning mechanisms into our tool Untestable Fault Omitter (UFO). The overall tool flow is given in Figure 5. For best performance, implications are computed for each node from inputs to outputs in a breadth-first manner (similar to [11]). As mentioned before, Extended Backward Learning of step 1(a) and our implication extraction techniques (section 3.1) of step 1(b) are performed together to minimize computation effort. After computing implications for a node, it checks to see if it satisfies the recurrence relation criteria; if so, it is marked as sequentially unachievable. Where as 1(a) and 1(b) are implications based on structural analysis, step 1(c) helps to learn sequentially unachievable nodes. More implications learnt from 1(a) or 1(b) can lead to finding more unacheivable values in 1(c). Once the implication graph is built, we compute the Phase I and Phase II of multi-node relations (section 3.2) in steps 2(a) and 2(b). All the learned information is then fed to the modified MUST procedure (explained below) in step 3, which includes both the stem analysis and untestable fault identification.

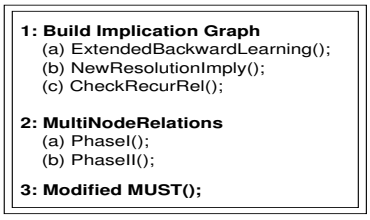


**Figure 5: Flow of UFO tool**

The original MUST algorithm of [8] does not employ an implication graph and it stores the necessary stem assignments for each fault as obtained in the pre-processing phase. Note that as more implications become available, the number of assignments needed to be stored can be huge and cause potential memory explosion. If an implication path is present in the circuit such as $X_v \rightarrow Y_v \rightarrow Z_v$ and if a fault *f* requires $Z_{\bar{v}}$ for its detection, then it also requires the nodes $Y_{\bar{v}}$ and $X_{\bar{v}}$. Storing only $Z_{\bar{v}}$ in the pre-processing stage is sufficient to obtain all other necessary assignments from the implication graph when fault *f* is targeted during the second phase of MUST. Our current implementation thus uses a depth-first traversal to go to the last node reached in a transitive closure and performs MUST for a node when it is encountered in the return path. Likewise, if there are strongly-connected components (SCCs) in the graph, phase I of MUST needs to be performed for only one representative node in the SCC to reduce computation costs. In our experiments, we found that the memory needed to store the necessary assignments was only two to three times the size of the implication graph for a circuit.

# 5. EXPERIMENTAL RESULTS

Experiments for ISCAS and ITC benchmark suites were conducted on a Pentium-4 2.6GHz machine having 1GB RAM and running Linux operating system. Table 1 shows the results. First, the maximum number of untestable faults identified among [3], [4], [5], [6], [7], [8], [10], [11], [12] and [13] are reported under column *Prev BEST*. Note that all of these works are *non branch-and-bound* based techniques, similar to ours. For each circuit, *#TF* shows the maximum implication learning depth set during our experiments. A depth of *n* means that implications are computed across *-n* to *+n* time-frames (*i.e.,* a total of *2n+1* frames). Then, we show the results of our UFO tool obtained using various procedures. *EBL Alone* reports the untestable faults identified using EBL implications and MUST procedure. *EBL + New SNI* reports the results obtained by using EBL and the proposed new single-node implications along with MUST, whereas *EBL + New SNI + MNR* reports the results obtained by using the multi-node relations extracted in addition to the single-node implications. The columns titled *Time*, *#SNI*, *#NEC*, *#MNR* and *#UNT* report the total time (static learning time + untestable fault identification time) in seconds, the number of single-node implications obtained, the sum of all necessary assignments obtained for all the remaining faults (faults not found to be untestable), the number of multi-node relations obtained, and the number of untestable faults obtained, respectively.

As shown in Table 1, the number of single-node implications obtained with our technique can be much more compared to those obtained from EBL alone. For example, for b07, the total number of implications increased from 54K to 189K because of our added learning. Similarly for s9234, the number of implications increased from 8.2M to 9.9M. Using all the newly learned relations, UFO was able to identify far more untestable faults with very low overhead. For smaller circuits, our tool identified considerably more faults except for s526 for which FUNI [5] could identify 62 untestable faults with the help of illegal states captured using BDDs. For larger circuits, UFO has shown tremendous improvement in many cases. For s13207.1, we were able to identify 1048 untestable faults where as the previous best was only 453. Similarly for the difficult circuit s38584.1, UFO identified 2431 untestable faults whereas only 1653 were identified among the previous works. For all the circuits, the maximum amount of memory needed by UFO was less than 17MB.

For small circuits, deterministic ATPG engines often perform better because of the smaller search-space. But for larger circuits, all the additional untestable faults identified by UFO were extremely difficult for *branch-and-bound* based engines to identify, as the numbers of untestable faults reported by deterministic ATPGs are much lower than ours. Besides, UFO was able to find significantly more number of necessary assignments for the remaining faults (shown in boldface) as compared to *EBL Alone*. These non-trivial necessary assignments obtained with low overheads indicate that our technique can be a very helpful pre-processor and can potentially speed up the sequential ATPG engines by pruning the search-space because of the increased necessary assignment set for the faults.

# 6. CONCLUSION

We have presented novel techniques based on powerful combinations of binary resolution and static logic implications to extract non-trivial relations from the circuits. These non-trivial relations include new single-node as well as multi-node implications. The multi-node relations proposed could be learned after any given depth of the recursive learning procedure as well. Experimental results show that our learning can enable the identification of significantly more untestable faults with low computational overhead and memory requirements. Finally, our tool can be employed as a preprocessor for a deterministic ATPG engine to first omit the identified untestable faults and use the necessary assignments for the rest of the faults as multiple objectives during test generation.

**Table 1: Experimental results for ISCAS and ITC Benchmarks**

| Circuit | Prev BEST | #TF | EBL Alone | | | | EBL + New SNI | | | | EBL + New SNI + MNR | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Time | #SNI | #NEC | #UNT | Time | #SNI | #NEC | #UNT | Time | #MNR | #NEC | #UNT |
| c432 | 2 | 0 | 0.02 | 2.4K | 11K | 2 | 0.03 | 2.4K | 11K | 2 | 0.03 | 132 | 11.5K | **4** |
| c2670 | 97 | 0 | 0.3 | 60K | 104K | 107 | 0.5 | 104K | 137K | **115** | 0.53 | 0.5K | **140K** | **115** |
| s298 | 21 | 2 | 0.2 | 10.7K | 33K | 13 | 0.24 | 14K | 36.3K | **25** | 0.4 | 2.8K | 39.5K | **25** |
| s386 | 63 | 2 | 0.51 | 33K | 84K | 67 | 0.72 | 36K | 87.3K | 69 | 0.89 | 7.4K | **94.8K** | **70** |
| s526 | **62*** | 2 | 0.55 | 20K | 80K | 12 | 0.67 | 22K | 86K | 24 | 0.93 | 7K | **92K** | 25 |
| s641 | **59** | 2 | 0.3 | 81K | 37K | **59** | 0.32 | 83K | 39K | **59** | 0.4 | 1K | 40K | **59** |
| s713 | **101** | 2 | 0.4 | 99K | 41K | **101** | 0.42 | 100K | 43K | **101** | 0.46 | 1.5K | 43.8K | **101** |
| s820 | 26 | 2 | 1.3 | 59K | 299K | 2 | 2.7 | 76K | 336K | 15 | 4.4 | 90K | **436K** | **33** |
| s832 | 28 | 2 | 1.46 | 58K | 301K | 11 | 2.8 | 75.8K | 339K | 25 | 4.8 | 96K | **441K** | **46** |
| s1238 | 58 | 2 | 1.2 | 73K | 261K | 53 | 1.93 | 75K | 259K | 59 | 3.4 | 29K | 272K | **62** |
| s1423 | **14** | 2 | 0.8 | 79K | 162K | **14** | 1.0 | 86K | 172K | **14** | 1.2 | 60 | 173K | **14** |
| s1494 | 15 | 2 | 5.95 | 151K | 778K | 10 | 9.85 | 164K | 791K | 10 | 12.3 | 126K | **899K** | **18** |
| s3330 | 494 | 2 | 5.6 | 1.51M | 629K | **495** | 9 | 1.52M | 639K | **495** | 15.5 | 32K | 647K | **495** |
| s4863 | 72 | 2 | 9.5 | 266K | 885K | 92 | 13.2 | 317K | 951K | **114** | 16.4 | 31K | **959K** | **114** |
| s5378 | 884 | 2 | 68 | 4.2M | 2.5M | **891** | 71.8 | 4.47M | 2.58M | **891** | 77.2 | 70K | **2.94M** | **891** |
| s9234 | 434 | 2 | 403 | 8.2M | 13.8M | 599 | 656 | 9.9M | 14.5M | 688 | 727 | 300K | **14.6M** | **733** |
| s9234.1 | 371 | 2 | 112 | 2.6M | 5.3M | 377 | 142 | 2.9M | 5.8M | **395** | 176 | 43K | **6.0M** | **395** |
| s13207 | 1125 | 2 | 333 | 16.2M | 7.88M | 1162 | 534 | 19.9M | 11.4M | 1206 | 640 | 300K | **12.9M** | **1220** |
| s13207.1 | 453 | 2 | 440 | 10.1M | 8.92M | 575 | 678 | 17.0M | 12.74M | 1008 | 776 | 90K | **14.7M** | **1048** |
| s15850 | 835 | 2 | 981 | 23.9M | 22.8M | 874 | 1052 | 26.9M | 26.2M | **891** | 1224 | 181K | **27.3M** | **891** |
| s15850.1 | 951 | 2 | 427 | 28.3M | 9.7M | 1221 | 482 | 29.0M | 10.8M | **1283** | 523 | 55K | **11.8M** | **1283** |
| s38417 | 511 | 2 | 2556 | 22.1M | 33.2M | 696 | 2923 | 23.9M | 35.3M | **760** | 3114 | 16.6K | **36.3M** | **760** |
| s38584 | 2283 | 1 | 2380 | 70.1M | 100M | 2662 | 2912 | 71.4M | 101.9M | 2672 | 3033 | 105K | 104.9M | **2681** |
| s38584.1 | 1653 | 1 | 2721 | 61.8M | 100M | 2416 | 2843 | 62.3M | 102.2M | 2423 | 2964 | 72K | 105.1M | **2431** |
| b07 | 2 | 2 | 2.50 | 54K | 174K | 9 | 6.3 | 189K | 484K | 32 | 7.2 | 125K | **498K** | **33** |
| b09 | 0 | 2 | 0.26 | 16K | 52K | 4 | 0.4 | 20K | 54K | **9** | 0.5 | 16K | **67K** | **9** |
| b10 | 1 | 2 | 0.36 | 15K | 60K | 1 | 0.7 | 26K | 63K | **14** | 1.2 | 15K | **92K** | **14** |
| b11 | 75 | 2 | 9.5 | 170K | 1M | 88 | 12.3 | 197K | 1.23M | 112 | 16 | 109K | **1.3M** | **116** |
| b12 | 0 | 2 | 10.8 | 520K | 1.57M | 8 | 17 | 688K | 1.9M | **11** | 23 | 291K | **2.2M** | **11** |
| b13 | 26 | 2 | 0.6 | 43K | 122K | 26 | 0.7 | 45.9K | 125K | **30** | 0.75 | 3K | 126K | **30** |
| b15 | 407 | 1 | 3498 | 13M | 59.2M | 600 | 4510 | 16.4M | 68.3M | 773 | 4923 | 250K | **70.8M** | **784** |

*Note (a): Prev BEST → Max #Unt faults among [3], [4], [5], [6], [7], [8], [10], [11], [12] and [13] (non branch-and-bound based techniques as ours)*
*(b): * → Obtained using FUNI [5] which finds illegal states using BDDs for identifying untestable faults*

# 7. REFERENCES


[1] V. D. Agrawal and S. T. Chakradhar, "Combinational ATPG theorems for identifying untestable faults in sequential circuits," In *IEEE Trans. on CAD of Integ'd Circuits and Sys.*, vol. 14, Sept. 1999, pp. 1155-1160.

[2] S. M. Reddy, I. Pomeranz, X. Lim and N. Z. Basturkmen, "New procedures for identifying undetectable and redundant faults in synchronous sequential circuits", In *Proc. of VLSI Test Symp.*, 1999, pp. 275-281.

[3] M. A. Iyer and M. Abramovici, "FIRE: a fault independent combinational redundancy algorithm", In *IEEE Trans. on VLSI Systems*, June 1996, vol 4, pp. 295-301.

[4] M. A. Iyer, D. E. Long, M. Abramovici, "Identifying sequential redundancies without search," In *Proc. of Design Automation Conf.*, 1996, pp. 457-462.

[5] D. E. Long, M. A. Iyer and M. Abramovici, "Identifying sequentially untestable faults using illegal states", In *Proc. of VLSI Test Symp.*, 1995, pp. 4-11.

[6] M. S. Hsiao, "Maximizing impossibilities for untestable fault identification", In *Proc. of DATE Conf.*, 2002, pp. 949-953.

[7] M. Syal and M. S. Hsiao, "Untestable fault identification using recurrence relations and impossible value assignments", In *Proc. of Int'l Conf on VLSI Design*, 2004, pp. 481-486.

[8] Q. Peng, M. Abramovici and J. Savir, "MUST: multiple-stem analysis for identifying sequentially untestable faults", In *Proc. of Int'l Test Conf.*, 2000, pp. 839-846.

[9] W. Kunz and D. K. Pradhan, "Recursive Learning: a new implication technique for efficient solutions to CAD problems-test, verification and optimization", In *IEEE Trans. on CAD of Integ'd Circuits and Sys.*, Sept 1994, vol 13, pp. 1149-1158.

[10] W. Cao and D. K. Pradhan, "Sequential redundancy identification using recursive learning", In *Proc. of Int'l Conf. Computer-Aided Design*, 1996, pp. 56-62.

[11] J. Zhao, M. Rudnick and J. Patel, "Static Logic Implication with application to fast redundancy identification", In *Proc. of VLSI Test Symp.*, 1997, pp. 288-293.

[12] J. Zhao, J. A. Newquist and J. Patel, "A graph traversal based framework for sequential logic implication with an application to c-cycle redundancy identification", In *Proc. of VLSI Design Conf.*, 2001, pp. 163-169.

[13] K. Gulrajani and M. S. Hsiao, "Multi-node static implications for redundancy identification", In *Proc. of Design, Automation and Test in Europe Conf.*, 2000, pp. 729-733.

[14] X. Lin, I. Pomeranz, and S. M. Reddy, "Techniques for improving the efficiency of sequential test generation", In *Proc. of Int'l. Conf. Computer-Aided Design*, 1999, pp. 147-151.

[15] M. H. Schulz and E. Auth, "Improved deterministic test pattern generation with applications to redundancy identification", In *IEEE Trans. on CAD of Integ'd Circuits and Sys.*, July 1989, vol 8, pp. 811-816.