6-1991

A Sixteen-Valued Algorithm for Test Generation in Combinational Circuits

Akhtar Uz Zaman

M. Ali

Carlos R.P. Hartmann
Syracuse University, chartman@syr.edu

SCHOOL OF COMPUTER
AND INFORMATION SCIENCE

Syracuse
University

SU-CIS-91-18

A Sixteen-Valued Algorithm for Test Generation in Combinational Circuits

Akhtar-uz-zaman M. Ali and Carlos R.P. Hartmann

June 1991

Suite 4-116
Center for Science and Technology
Syracuse, New York 13244-4100

SU-CIS-91-18

A Sixteen-Valued Algorithm for Test Generation in Combinational Circuits

Akhtar-uz-zaman M. Ali and Carlos R.P. Hartmann

June 1991

School of Computer and Information Science
Suite 4-116
Center for Science and Technology
Syracuse, New York 13244-4100

(315) 443-2368

Syracuse University

School of Computer and Information Science

Technical Report No. SU-CIS-91-18

A Sixteen-Valued Algorithm for Test Generation in Combinational Circuits[1]

Akhtar-uz-zaman M. Ali[2]

Carlos R. P. Hartmann[3]

Abstract A 16-valued logic system for testing combinational circuits is presented. This logic system has been used to develop SIMPLE, an efficient test generation algorithm for single stuck-at faults.

The proposed scheme for testing stuck-at faults is based on imposing all the constraints that must be satisfied in order to sensitize a path from a fault site to a primary output. Consequently all deterministic implications are fully considered prior to the enumeration process. The resulting ability to identify inconsistencies prior to enumeration improves the possibility of quicker identification of redundant faults. In order to prune the search space we have introduced several speed-up techniques that effectively combine the information provided by the deterministic path sensitization and that obtained from the circuit topology.

Some properties of undetectable faults are presented and methods to identify them without actual test generation are proposed.

---

[1]This work was partially supported by the Rome Air Development Center under Contract No. F30602-88-D-0027 and a Senate Research Grant from Syracuse University.

[2]A. M. Ali was with the Department of Electrical & Computer Engineering, 121 Link Hall, Syracuse University, Syracuse, NY 13244-1240. He is now with IBM Corporation, Neighborhood Road, Dept. 32UA, Mail Station 467, Kingston, NY 12401. (e-mail: ali@kgnvmf.vnet.ibm.com)

[3]C. R. P. Hartmann is with the School of Computer and Information Science, Suite 4-116, Center for Science and Technology, Syracuse University, Syracuse, N.Y. 13244-4100. (e-mail: hartmann@top.cis.syr.edu)

Contents

List of Tables

List of Figures

1 Introduction

The generation of test patterns for combinational circuits has been long recognized by researchers as a well defined mathematical problem which belongs to the class of NP-complete problems [12, 15]. Several Automatic Test Pattern Generation (ATPG) algorithms for detecting stuck-at-faults in combinational circuits exist in the literature [8, 9, 11, 13, 14, 17, 20, 21, 22]. Most researchers characterize test pattern generation as a search problem and address strategies to make this search process efficient. For realistic circuit sizes the search space is prohibitively large and, to make matters worse, a solution is not always guaranteed to exist. However, as PODEM [14] first demonstrated, it is not necessary to explicitly search the entire space — sometimes a partial search can determine a test pattern or the fact that none exists. In fact the huge amount of backtracking computation that is sometimes required before recognizing that a test cannot be generated for a particular fault (such faults are termed *redundant* faults) proves to be a major bottleneck in any ATPG algorithm. In order to overcome this difficulty different strategies have been developed by researchers. These strategies vary from making use of unique implications to using circuit topology information. In spite of the improvements achieved by these strategies test pattern generation still remains a complex problem and the possibility of further improvements a viable one.

In this report we propose a 16-valued logic system and show its usefulness in combinational circuit testing. In § 2 we present an ATPG algorithm called SIMPLE that is based on this 16-valued logic system. This gate level algorithm uses a single stuck-at fault model and includes XOR and XNOR as primitive gates. In fact we will show how the proposed logic system can exploit the linearity of the XOR/XNOR gates. The key feature in SIMPLE is the derivation of all deterministic constraints that must be satisfied for propagating sensitization along a chosen path. The use of necessary assignments coupled with the completeness of the logic system helps in pruning the search space.

To improve the performance of SIMPLE we present, in § 3, several speed-up techniques. These are dynamic in nature and are based on the circuit topology and the information derived by the test generation process. We will show we can identify several independent subcircuits so that the value justification problem can be effectively divided into smaller problems.

The existence of undetectable faults is one of the main factors that cause test generation to be a complex procedure. With this in mind we present, in §4, methods to identify undetectable faults. These will be based on using information about already determined undetectable faults to identify newer ones and also on using circuit topology to identify undetectable faults without actual test generation.

2 SIMPLE: An ATPG Algorithm

In this section we present SIMPLE (**SI**xteen valued, **M**aximized **P**ropagation **L**owered **E**numeration approach to test generation), an ATPG algorithm for detecting single stuck-at-faults in combinational circuits that contain NOT, AND, NAND, OR, NOR, XOR and XNOR gates. This algorithm is based on a 16-valued logic system and introduces some novel approaches to make test pattern generation more efficient. There are three distinct phases in the algorithm presented here:

(*i*) **Pre-processing phase** (§2.2): In this phase we construct a set of trees based on the interdependence of circuit nets. Among other things this forest will be used to easily identify which circuit nets *must* be sensitized to derive a test. We also compute the token vectors which keep track of the parity of inversions between nets. This information is useful because it can identify which inputs of a gate may or may not be simultaneously sensitized.

(*ii*) **Propagation phase** (§2.3): In this phase we deliberately sensitize a single path (say $\boldsymbol{p_i}$) from the fault site (say $\boldsymbol{f}$) to a primary output (PO) and find all the resulting deterministic forward and backward implications. In the process other paths may get sensitized. Path selection is the only choice made in this phase—implications are based on all the constraints that *must* be satisfied in order to sensitize the chosen path. This is possible because of the completeness of the logic system and the use of deterministic implication rules. In a manner analogous to the D-algorithm [21] we use test cubes whose entries reflect the current values of all nets during any stage of test generation. The test cube obtained after all deterministic implications have been performed will be denoted as $\boldsymbol{T_f(p_i)}$.

(*iii*) **Enumeration phase** (§2.4): In general, the test cube $\boldsymbol{T_f(p_i)}$ constructed by the propagation phase may not yield a test—particularly because no arbitrary choices were made in the assignment of net values. Thus there may be gates whose input net values contain combinations capable of desensitizing the chosen path. In this phase we use an enumeration procedure to choose values for the primary inputs (PIs) so that such combinations can never occur.

In § 3 we will discuss how the proposed algorithm can be further improved by the incorporation of several speed-up techniques.

2.1 Preliminaries

2.1.1 The Logic System Used

Test generation involves considering the value of a net in the good and the faulty circuit. This can be done by representing the value of a net as an ordered pair (b_g, b_f) where $b_g(b_f)$

is the value of the net in the good (faulty) circuit [18]. Using this representation the value of a net can be one of the elements of the set $\boldsymbol{U}= \{(0,0),(0,1),(1,0),(1,1)\}$. Thus a net which has a different logic value in the good and the faulty circuit can either have the value (1,0) or (0,1). Conventionally these values have been referred to as $\boldsymbol{D}$ and $\overline{\boldsymbol{D}}$ respectively [21]. Thus any test for a *s-a-*1 (or *s-a-*0) fault must cause the value of the faulty net to be $\boldsymbol{D}$ or $(\overline{\boldsymbol{D}})$. Consequently the test generation process is initialized by introducing this "difference" between the good and the faulty circuit in some form e.g. the primitive D-cube of the failure (***pdcf***) [21], initial objective [14] etc. The initialization procedure in our algorithm is based on the concept of *fault site testing* which tries to generate the conditions that must be satisfied in order to test a particular fault site without imposing the constraint of a particular stuck-at fault at the site. In the next subsection we will show how we can take into account the common propagation requirements of the two stuck-at faults at the same net. We do this by introducing a new variable which contains the information that there is a difference between the normal and faulty circuits without imposing any constraints about the direction of the difference — this variable is denoted by $\boldsymbol{\Delta}$. So $\boldsymbol{\Delta}= \{(x,\overline{x})\}$ and the corresponding $\overline{\boldsymbol{\Delta}}= \{(\overline{x},x)\}$ where $x \in \{0,1\}$. Thus our algorithm is initialized by setting the value of the fault site to be tested to $\boldsymbol{\Delta}$. In the process of generating tests it might not be possible to uniquely specify the value of a net as one of the elements of $\boldsymbol{U}$. However, we may already know that a net cannot assume one or more of these values. We incorporate this information by defining the value of a net as one of the 16 subsets of $\boldsymbol{U}$. We denote these 16 sets as $\boldsymbol{\phi}$, $\mathbf{0}$, $\mathbf{1}$, $\boldsymbol{\Delta}$, $\overline{\boldsymbol{\Delta}}$, $\mathbf{0/1}$, $\mathbf{0}/\boldsymbol{\Delta}$, $\mathbf{1}/\boldsymbol{\Delta}$, $\mathbf{0}/\overline{\boldsymbol{\Delta}}$, $\mathbf{1}/\overline{\boldsymbol{\Delta}}$, $\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$, $\mathbf{0/1}/\boldsymbol{\Delta}$, $\mathbf{0/1}/\overline{\boldsymbol{\Delta}}$, $\mathbf{0}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$, $\mathbf{1}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$, and $\mathbf{0/1}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$ where $\mathbf{0}=\{(0,0)\}$, $\mathbf{1}=\{(1,1)\}$, $\boldsymbol{\Delta}=\{(x,\overline{x})\}$, $\overline{\boldsymbol{\Delta}}= \{(\overline{x},x)\}$ and "/" denotes set union. Note that $\boldsymbol{U}=\mathbf{0/1}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$. The value $\boldsymbol{\phi}$ needs to be included to reflect the situation when two or more constraints require disjoint values at any net. If we set $\boldsymbol{\Delta}= \boldsymbol{D}$ (or $\overline{\boldsymbol{\Delta}}= \boldsymbol{D}$) then these 16 values would be equivalent to the elements of the logic system developed by Akers [3] to provide a tool for test generation. Tables 1, 2 and 3 represent the AND, NOT, and XOR functions in our 16-value system for the values $\mathbf{0}$, $\mathbf{1}$, $\boldsymbol{\Delta}$, and $\overline{\boldsymbol{\Delta}}$. The complete table for all the 15 non-$\boldsymbol{\phi}$ values can be easily constructed from the given tables by using the set union operation. The tables for all other logic functions can be obtained from these three tables. Note that any logic function with $\boldsymbol{\phi}$ as one of its arguments will yield $\boldsymbol{\phi}$ as a result. Using this notation we will define a sensitized net as one whose value is either $\boldsymbol{\Delta}$, $\overline{\boldsymbol{\Delta}}$, or $\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$. Furthermore, if all the nets along a path in the circuit are sensitized, then the path is said to be sensitized. As will be seen later on, this 16-valued system exploits the linearity of XOR/XNOR gates during test generation. It also allows us to characterize all restrictions that are imposed by a fault and the particular circuit path chosen in order to propagate its effect.

AND	0	1	Δ	$\overline{\Delta}$
0	0	0	0	0
1	0	1	Δ	$\overline{\Delta}$
Δ	0	Δ	Δ	0
$\overline{\Delta}$	0	$\overline{\Delta}$	0	$\overline{\Delta}$

Table 1: AND Table

Variable	0	1	Δ	$\overline{\Delta}$
Complement	1	0	$\overline{\Delta}$	Δ

Table 2: NOT Table

XOR	0	1	Δ	$\overline{\Delta}$
0	0	1	Δ	$\overline{\Delta}$
1	1	0	$\overline{\Delta}$	Δ
Δ	Δ	$\overline{\Delta}$	0	1
$\overline{\Delta}$	$\overline{\Delta}$	Δ	1	0

Table 3: XOR Table

2.1.2 Fault Site Testing

In this section we discuss how we can exploit the common requirements that are imposed when we sensitize the same path from the fault site to a PO in order to generate tests for both stuck-at faults at this net. In order to do this we cannot impose the conditions required to sensitize the fault site until the common requirements are taken into consideration. As discussed earlier we must introduce the "difference" between the normal and faulty circuit without imposing the constraint about the direction of the difference. We do this by introducing a fictitious gate G_f at the site of the fault. If the fault is at net n we introduce G_f between net n and a newly created net n_f. Net n_f is connected to all nets which were previously connected to n. In order to take into account both stuck-at faults at a given net our initial test cube should have the following values for nets n and n_f:

n	n_f
0/1	$\boldsymbol{\Delta}$

We will then execute the Propagation Phase of the algorithm (§2.3) to sensitize a path $\boldsymbol{p_i}$ from net n_f to some PO. This phase will impose all the deterministic constraints of propagating sensitization along path $\boldsymbol{p_i}$. As mentioned earlier, the resulting test cube obtained at the end of the Propagation Phase will be denoted as $\boldsymbol{T_f(p_i)}$. Since $\boldsymbol{T_f(p_i)}$ does not take any particular stuck-at fault into account we then construct two test cubes by setting the value of net n to **0(1)** and find its corresponding deterministic implications to generate $\boldsymbol{T_{f_1}(p_i)}$ ($\boldsymbol{T_{f_0}(p_i)}$) for an s-a-1 (s-a-0) fault at net n. The Enumeration Phase can then be independently executed for both these deterministic test cubes in order to generate tests for both the faults. This procedure and its merits will be made clearer later with the help of an example. As opposed to fault site testing, conventional fault testing for a stuck-at fault at a net n would require the following initialized values:

		n	n_f
n	s-a-0	**1**	$\boldsymbol{D}$
n	s-a-1	**0**	$\overline{\boldsymbol{D}}$

2.1.3 Deriving Common Requirements for Testing Different Checkpoints

It is well known that a test set, that detects all single stuck-at faults at the PI nets, fanout branch nets and the output nets of all XOR/XNOR gates of a circuit, will detect all single stuck-at faults in the circuit [6]. Thus these nets, henceforth referred to as "generalized checkpoints," constitute our initial list of target faults for which tests have to be generated. However, if any of these target faults is undetectable, additional target faults must be considered [1, 10, 19].

In this section we investigate the possibility of reducing the computation required in testing several checkpoints by first considering their common requirements and performing this computation only once.

Consider a two-input AND gate G, shown in Figure 1(a), where both inputs of G are generalized checkpoints and thus belong to our initial list of target faults. Instead of testing each of the inputs separately, we first impose the constraints that must be satisfied to site-test the output net of G as shown in Figure 1(b). Figure 1(c) shows the additional constraints that must be imposed on the test cube obtained at the end of the Propagation Phase in order to generate tests for all four faults at the inputs of G.

The above procedure should be adopted whenever we encounter a gate which has at least two inputs belonging to the set of generalized checkpoints. We would then perform the Propagation Phase of the algorithm by considering the output of the gate as the fault site. Once this is successfully done the Enumeration Phase can be performed independently for each of the checkpoint faults. In the situation that any of these checkpoint point faults cannot be detected by sensitizing the chosen path then alternate paths, if any, must be investigated for this fault.

2.2 Pre-processing Phase

To illustrate the various phases of our algorithm we will consider net 25, of the example circuit shown in Figure 2, as the fault site. Note that nets 22 and 23 belong to the set of generalized checkpoints and thus belong to our initial list of target faults. Hence we will consider net 25 as the fault site and perform the sensitization of a path. Consequently we introduce the gate G_f whose input and output are nets 25 and 25_f respectively.

2.2.1 Construction of Dominator Forest

The importance of identifying nets that *must* be sensitized for a fault to be detected was first highlighted by Akers [3] and later by Fujiwara and Shimono [11]. As pointed out in TOPS [17], the concept of graph dominators [23] can be used to identify the nets which *must* be sensitized to detect a fault. In the context of test generation we term the set of dominators of a net m as the set of all nets in the circuit which lie on every path from net m to any PO. By definition, net m is a dominator of itself; however, for ease of notation we define $D(m)$ as the set of all dominators of m except m itself. To account for multiple output circuits the concept of dominator tree can be extended to that of a forest. We present here a procedure to construct this forest for a given circuit. This forest will not only be used to compute the dominators for a particular fault site; but also for the sensitization of subpaths, selection of

PIs in the Enumeration Phase and generation of the initial list of target faults.

We construct a set of trees such that every net of the circuit corresponds to a node in one of the trees in the forest. We start by creating as many trees as there are POs such that each PO corresponds to a root of a tree. However, new trees may be created during the procedure. Thereafter, each node which has not been marked as a leaf is inspected and the tree construction is continued as follows:

(i) If the node m_i being considered corresponds to the output net of a logic gate G in the circuit, then every input net of G becomes a child of this node m_i. Furthermore, if the input net is a PI it is marked as a PI leaf. If the input net is a fanout branch (FOB), then it is marked as a FOB leaf.

(ii) If the node m_i being inspected is a fanout stem (FOS), then wait until all the FOBs corresponding to this FOS have been marked as FOB leaves. Then find the immediate common ancestor of all these FOB leaves. If such an ancestor exists, then make m_i a child of this ancestor node. If it does not, then start a new tree with m_i as a root. In either case, mark m_i as an FOS node—if it is also a PI, then it must be marked as a PI leaf also.

The above procedure is continued until every net of the circuit becomes a node in some tree of the forest.

The forest construction is based on the following properties:

1. The dominance relation is transitive

2. If a FOS net m_i has n_i FOB nets denoted by $m_{i1}, m_{i2}, \ldots, m_{in_i}$, then

   $$D(m_i) = \bigcap_{j=1}^{n_i} D(m_{ij})$$

3. The output net of any gate G is a dominator for every input net of G

The root of any tree in the constructed forest is either a PO or a FOS. If any tree has a single node, then this node must either correspond to a PI which is also a FOS net or a PO which is also a FOB net. The leaves of the trees in the forest correspond to the *checkpoints*, i.e., the PIs and the FOBs. Thus our initial list of target faults consists of all leaves of the trees of the dominator forest and the output of all XOR/XNOR gates [6]. However, as pointed out earlier, in case any of these target faults are undetectable additional target faults must be considered [1, 10, 19].

The set $D(m)$ contains all the nodes encountered when traversing the tree (in which m is a node) from m to the root. Recall that $m \notin D(m)$.

The "basis nodes," as defined in TOPS [17], can also be identified easily from the domi-

nator forest.[1] However, keeping in mind that a node cannot be a basis node unless all FOS nets that influence it have completely reconverged prior to it, we adopt a simpler approach of identifying which nodes are NOT basis nodes. Thus, instead of inspecting each node to verify whether it is a basis node or not, we pick one FOS net at a time to generate the set of nodes which are NOT basis nodes. Let there be k FOS nets denoted by m_i, $i = 1, 2, \ldots, k$. Furthermore, let the FOS net m_i have n_i FOB nets denoted by $m_{i1}, m_{i2}, \ldots, m_{in_i}$. The set of nodes which are NOT basis nodes is given by

$$\bigcup_{i=1}^{k} \left[\bigcup_{j=1}^{n_i} [D(m_{ij}) \cup \{m_{ij}\}] - D(m_i) \right].$$

To prove the above assertion consider a net $m_\ell \in \bigcup_{i=1}^{k} \left[\bigcup_{j=1}^{n_i} [D(m_{ij}) \cup \{m_{ij}\}] - D(m_i)\right]$. Thus there must exist i and j, $1 \leq i \leq k$ and $1 \leq j \leq n_i$, such that $m_\ell \in D(m_{ij}) \cup \{m_{ij}\}$ and $m_\ell \notin D(m_i)$. In other words m_ℓ is influenced by a FOS net m_i all of whose fanout branches do not reconverge prior to net m_ℓ. Thus m_ℓ is not a basis node. Conversely if m_ℓ is not a basis node then it must be influenced by some FOS net(s) all of whose fanout branches do not reconverge prior to net m_ℓ. Tracing back paths from net m_ℓ to the PIs let m_i $(1 \leq i \leq k)$ be the first such FOS net (i.e. those whose branches do not reconverge prior to net m_ℓ) encountered. If there are any other FOS nets between m_i and m_ℓ then they must totally reconverge prior to m_ℓ. Thus there must be a FOB net m_{ij} $(1 \leq j \leq n_i)$ corresponding to the FOS net m_i such that $m_\ell \in D(m_{ij}) \cup \{m_{ij}\}$ and $m_\ell \notin D(m_i)$. Thus $m_\ell \in \bigcup_{i=1}^{k} \left[\bigcup_{j=1}^{n_i} [D(m_{ij}) \cup \{m_{ij}\}] - D(m_i)\right]$.

Explicit evaluation of the above expression is, however, not necessary. We can keep track of the basis nodes while constructing the dominator forest. Recall that we have to identify the immediate common ancestor of all the FOB nets corresponding to a FOS net in order to determine the position of the latter in the forest. If such an ancestor exists then all nets, excluding this common immediate ancestor, that are encountered when traversing the trees from every FOB net to the immediate ancestor belong to the set of NOT basis nodes. If such an ancestor does not exist, then all nets encountered when traversing the trees from every FOB net to the root of its tree belong to the set of NOT basis nodes. Note that in either case all the FOB nets are also included in this set. Consequently, all nodes not belonging to this set are basis nodes. Furthermore the justification of a **0** or a **1** at these nodes will not lead to contradictions provided there is no net in the good circuit which has a constant value independent of the PIs.

The dominator forest for the circuit in Figure 2 is shown in Figure 3. Note that the only basis nodes for this circuit are the PIs.

[1] A net, say m, is defined to be a basis node if and only if all FOS nets that influence m totally reconverge prior to it [17].

2.2.2 Token Assignment

The goal of this stage is to identify which circuit nets can or cannot be affected by a fault. In order to convey this information we associate with every net a Boolean token. This token will be TRUE if and only if there exists a path from n_f to any PO which passes through this net. These tokens can be computed by a single forward pass through the circuit. In the example we are considering the nets which are assigned a TRUE token are 25_f, 26–28, 33–35, 39, 40 and 42–52. In a later section we will extend the concept of a Boolean token to that of a token vector which will be useful if there are XOR/XNOR gates on the path being sensitized.

2.3 Propagation Phase

In this phase we sensitize a single path from net n_f to a PO, however, other paths may also get sensitized. As mentioned before we use test cubes whose entries reflect the current values of all nets during any stage of test generation. The entries of any test cube, $\boldsymbol{tc}_k$, are elements of our 16-valued system.

We initialize this phase by constructing $\boldsymbol{tc}_1$ in the following manner:

1. Set net n_f to the value $\boldsymbol{\Delta}$.
2. Assign $\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$ to all nets belonging to the set $D(n)$.
3. Set all nets with FALSE tokens to $\mathbf{0/1}$.
4. Assign $\mathbf{0/1}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$ to all unassigned nets of the test cube.

In our example $D(25) = \{48\}$, and the resulting $\boldsymbol{tc}_1$ is shown below where only nets whose entries are different from $\mathbf{0/1}$ and $\mathbf{0/1}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$ are shown.

25_f	48
$\boldsymbol{\Delta}$	$\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$

All other nets either have the value $\mathbf{0/1}$ (if they have a FALSE token) or the value $\mathbf{0/1}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$ (if they have a TRUE token).

For each test cube $\boldsymbol{tc}_k$ generated at any stage of our algorithm we find its corresponding "deterministic" test cube, $\boldsymbol{d}(\boldsymbol{tc}_k)$. We define a $\boldsymbol{d}(\boldsymbol{tc}_k)$ as one in which no entry can be changed without making some arbitrary choice(s) in one or more net values. That is, all unique implications of the net values must be considered. Rules for forward and backward implication procedures to be used in constructing $\boldsymbol{d}(\boldsymbol{tc}_k)$ from $\boldsymbol{tc}_k$ are given in Appendix A. If in any $\boldsymbol{d}(\boldsymbol{tc}_j)$ we have a sensitized path $\boldsymbol{p}_i$ from the fault site to any PO, then the Enumeration

Phase is invoked. This test cube, $d(tc_j)$, is denoted as $T_f(p_i)$. As mentioned earlier, we must construct $T_{f0}(p_i)$ and $T_{f1}(p_i)$ from $T_f(p_i)$ in order to derive tests for the s-a-0 and s-a-1 faults at net n respectively.

The $d(tc_1)$ for our example is shown below. Only the entries for nets whose values are different from those in tc_1 are listed. In fact, for each cube for the example we are considering only the entries whose values are different from those in the preceding one will be shown.

26	27	28	33	34	35	39	40
Δ	Δ	Δ	$0/\Delta$	$0/\overline{\Delta}$	$0/\Delta$	$0/1/\Delta$	$0/1/\overline{\Delta}$

41	47	49	50	51	52
0	$0/\Delta/\overline{\Delta}$	$\Delta/\overline{\Delta}$	$\Delta/\overline{\Delta}$	$0/\Delta/\overline{\Delta}$	$1/\Delta/\overline{\Delta}$

If $d(tc_1)$ cannot be constructed because contradictions were encountered, then there exists no test for the two stuck-at faults at net n. If during the construction of $d(tc_1)$ the value of net n changes from $0/1$ to 0 (1) then there is no test for the fault net n s-a-0 (s-a-1). If $d(tc_1)$ is successfully constructed then we have a sensitized path from n_f to all the FOB nets corresponding to the first FOS node (could be n itself!) encountered in traversing the appropriate tree of the dominator forest from n to the root. If there is no FOS encountered, then we have a sensitized path from n_f to the PO corresponding to the root of the tree. In our example, we have sensitized paths till the FOB nets 26, 27 and 28.

At this point we have to select one of the FOB nets, say m_1, to extend the sensitized path. The use of testability measures should be incorporated into this selection process in order to make it more efficient. The discussion in this section will, however, be kept general and no specific heuristic will be referred to. To obtain tc_2 we should sensitize all nets belonging to the set $D(m_1) - D(n)$ by intersecting their values in $d(tc_1)$ with $\Delta/\overline{\Delta}$. If any empty intersection results, then the sensitized path cannot be extended through m_1 and alternate paths should be investigated. Note that this step is implicitly performing the equivalent of the X-path check [14] while setting up which gate outputs should be sensitized. As stated earlier, we would then construct $d(tc_2)$. If contradictions occur while constructing $d(tc_2)$, then an alternate path must be selected. Otherwise we have a sensitized path from n_f to at least the FOB nets corresponding to the next FOS net or some PO. Assume that we extend the sensitized path in our example through net 28. We use $D(28) - D(25) = \{35\}$ to construct the tc_2 and $d(tc_2)$ shown below. We now have sensitized paths till the FOB nets 49 and 50.

tc_2:

35
Δ

$d(tc_2)$:

13	29	30	36	47	48	49	50	51	52
1	1	1	1	0/Δ	Δ	Δ	Δ	0/Δ	1

The process of extending the sensitized path by selecting a FOB net, constructing a tc_k and its corresponding $d(tc_k)$ is continued until we reach some PO and have constructed $T_f(p_i)$. If contradictions occur, then alternate paths should be investigated. If all possible paths give contradiction, then no test exists. Note that all possible single paths need not be explicitly investigated to arrive at this conclusion—for example, if all paths from net n to any net $m \in D(n)$ gives contradictions, then we can conclude that no test exists. If during the construction of $d(tc_k)$, $k \geq 2$, the value of net n changes from **0/1** to **0** (**1**) then an alternate path, if one exists, must be investigated to derive a test for the fault net n s-a-0 (s-a-1). Proceeding with our example, let us extend the sensitized path through net 49. (Note that the attempt to extend the sensitized path through net 50 would lead to a contradiction and this would be identified immediately from the existing and required value of net 52.) Thus we use $D(49) - D(48) = \{51\}$ in order to construct the tc_3 and $d(tc_3)$ shown below. We now have a sensitized path (say p_1) from 25_f to a PO, and thus $d(tc_3)$ is $T_f(p_1)$.
tc_3:

51
Δ

$d(tc_3)$:

1	17	18
1	1	1

At this point we have to set the value net 25 to **0**(**1**) and find its corresponding deterministic implications to generate $T_{f1}(p_i)$ ($T_{f0}(p_i)$) respectively. We illustrate this process by considering only the s-a-0 fault at net 25. Thus we set the value of net 25 to **1** and construct the corresponding deterministic test cube shown below:
$T_{f0}(p_i)$:

9	10	21	22	23	24
1	1	1	1	1	1

In order to keep the discussion as general as possible, we will henceforth use the notation $T_{fv}(p_i)$, where $v \in \{0, 1\}$, instead of making specific references to $T_{f1}(p_i)$ or $T_{f0}(p_i)$.

Note that all the constraints imposed by $T_{fv}(p_i)$ must be satisfied in order to sensitize path p_i. Since the backward implication rule does not make any arbitrary choices, there

may be gates where the output value is a proper subset of the value implied by the input values, i.e., the input values include combination(s) that will desensitize path $\boldsymbol{p_i}$. In view of this fact we introduce the following definition. If, in a deterministic test cube $\boldsymbol{d(tc_k)}$, the value of the output net m of a gate G is a proper subset of the value implied at net m by the input values, in $\boldsymbol{d(tc_k)}$, of G then net m is said to be a **variant** net in $\boldsymbol{d(tc_k)}$. If a net is not variant it is defined to be **invariant** in $\boldsymbol{d(tc_k)}$. In our example, nets 41 and 47 are the variant nets.

If all the nets in the circuit are invariant nets in $\boldsymbol{T_{fv}(p_i)}$ then the specified PIs in $\boldsymbol{T_{fv}(p_i)}$ represent all the requirements that must be satisfied by any input pattern that detects the fault net n_f *s-a-v* by sensitizing path $\boldsymbol{p_i}$. In general, however, not all nets in $\boldsymbol{T_{fv}(p_i)}$ will be invariant. In such a situation there exists an assignment of the unspecified PIs (i.e., inputs with the **0/1** value) in $\boldsymbol{T_{fv}(p_i)}$ which will desensitize path $\boldsymbol{p_i}$. In order to obtain a test from $\boldsymbol{T_{fv}(p_i)}$ we must convert all variant nets to invariant ones by specifying one or more of these PIs. Moreover, the new deterministic test cube obtained by specifying these PIs in $\boldsymbol{T_{fv}(p_i)}$ should result in net values that are subsets of their corresponding values in $\boldsymbol{T_{fv}(p_i)}$ for all the nets of the circuit. This condition is required to prevent the setting of PIs in such a way as to result in a disallowed value at a net that was variant in $\boldsymbol{T_{fv}(p_i)}$. For example, if we set PI nets 15 and 16 to the value **1** in the $\boldsymbol{T_{fo}(p_i)}$ obtained, then the resulting deterministic test cube would have the value **1** at net 47 and also at both the POs of the circuit.

We re-emphasize that conversion of variant nets to invariant ones will always involve some arbitrary choice(s). Different approaches can be adopted to make choices that will convert all variant nets in $\boldsymbol{T_{fv}(p_i)}$ to invariant ones with values that are subsets of the corresponding net values in $\boldsymbol{T_{fv}(p_i)}$ for all the nets of the circuit, provided there exists an input pattern that sensitizes path $\boldsymbol{p_i}$. In this report we will describe an enumeration approach and a consistency checking approach. However only the enumeration procedure will be used in our example.

2.4 Enumeration Phase

The goal of this phase is to obtain a test by specifying the unassigned PIs in $\boldsymbol{T_{fv}(p_i)}$ such that all nets are invariant in the resulting deterministic test cube and have values that are subsets of their corresponding values in $\boldsymbol{T_{fv}(p_i)}$. In order to convert variant nets into invariant ones we assign values to different PIs such that the resulting value at the net is a subset of the required value. In § 3 we will discuss how we can prioritize the value justification of the variant nets and how we can guide the selection of the PIs to which values should be assigned.

We start by choosing an unspecified PI, I_{l_1}, in $\boldsymbol{T_{fv}(p_i)}$ and assign a logic value (**0** or **1**) to it, thereby creating a new test cube which we denote by $\boldsymbol{tc_{fv}(p_i, 1)}$. Now we find its

corresponding deterministic test cube $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \mathbf{1}))$ and update its list of variant nets (note that new variant nets may be created). However if $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \mathbf{1}))$ cannot be obtained due to some contradiction, then we complement the entry for I_{l_1} in $\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \mathbf{1})$ and construct its corresponding $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \mathbf{1}))$. If this also leads to a contradiction, then there exists no test corresponding to $\boldsymbol{T}_{fv}(\boldsymbol{p}_i)$. If we are successful in constructing $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \mathbf{1}))$, we assign a logic value to some other unspecified PI, I_{l_2}, thereby creating $\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \mathbf{2})$. As before we must construct $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \mathbf{2}))$ and update its list of variant nets. This procedure is continued and we traverse the decision tree, in a manner analogous to PODEM [14], until one of the following two conditions occur:

- The list of variant nets corresponding to some $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \boldsymbol{j}))$ becomes empty.
- The decision tree is exhausted, i.e. no test exists.

If the procedure is terminated because the former condition is satisfied, then the values of the PIs in $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \boldsymbol{j}))$ represent test(s) for the fault. To derive test patterns for the fault we would then assign either **0** or **1** to those PIs in $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \boldsymbol{j}))$ which have the value $\mathbf{0/1}$.

Heuristics like controllability measures should be used in this phase to guide the selection of the PIs to be assigned values. However, as in the case of the Propagation Phase we will avoid making reference to any particular measure in order to preserve generality.

Returning to our example, nets 41 and 47 are the only variant nets in $\boldsymbol{T}_{fo}(\boldsymbol{p}_1)$. By inspecting the dominator forest we notice that nets 15 and 16 are the PIs which are "closest" to net 47. We thus start by setting net 15 to **0**—however, this does not change the value of any other net. We continue by setting net 16 to **0**—once again no new changes result. We now use the dominator forest to reach the FOS net 42 and thus determine that nets 8 and 12 are the next "closest" PIs. We could, for example, set net 8 to **0**—the only resulting change is a $\mathbf{0}/\boldsymbol{\Delta}$ at net 39. We then set the value of net 12 to **0**—this changes the value of net 40 to $\mathbf{0}/\overline{\boldsymbol{\Delta}}$. Continuing in this fashion we would set net 7 to **0**—this would result in a **0** at nets 33, 39 and 47 and a $\mathbf{0}/\overline{\boldsymbol{\Delta}}$ at nets 42 through 46. However net 47 is still a variant net and we continue the enumeration process. If we now set net 11 to a **0** it will result in a $\overline{\boldsymbol{\Delta}}$ at 34, 40 and 42 through 46; it would also give a contradiction at net 47. So we go back and change the value of the last PI that was assigned i.e. net 11 is now set to a **1**. This will result in a **0** at nets 34, 40 and 42 through 47. Thus net 47 has been converted to an invariant one. A similar procedure can be followed to covert net 41 to an invariant one. For example, setting nets 2, 3 and 4 to the value **1** would achieve the required conversion.

Note that the generation of tests for the *s-a*-1 faults at the checkpoint nets 22 and 23, while sensitizing the same path $\boldsymbol{p}_1$, involves only the construction of $\boldsymbol{T}_{f_1}(\boldsymbol{p}_1)$ with appropriate

values for nets 22 and 23 and then executing the Enumeration Phase. In summary, we have used the same $T_f(p_1)$, which is $d(tc_3)$ of our example, to obtain tests for four single stuck-at faults at two checkpoint nets. These are the faults net 22 s-a-0, net 22 s-a-1, net 23 s-a-0 and net 23 s-a-1.

Since the conversion of variant nets to invariant ones is the key to generating a test from $T_{fv}(p_i)$ it is useful to keep track of nets which are variant in the process of constructing $T_{fv}(p_i)$. This would avoid the unnecessary checking of every net as variant or invariant after $T_{fv}(p_i)$ has been constructed. Note that if a net is invariant at some stage of generating a test for a fault it will not become variant unless a new backward implication (with a value which is a proper subset of the existing value) is made for the net.

The algorithm described so far can be substantially improved by the introduction of several speed-up techniques which we discuss in §3.

2.5 Consistency Checking Approach

In this section we discuss an alternate procedure that can be used instead of the Enumeration Phase in order to convert all the variant nets in $T_{fv}(p_i)$ to invariant ones. The approach followed here is similar to the consistency operation of the D-algorithm [21]. Recall that a gate whose output is a variant net is characterized by having inputs, specified in $T_{fv}(p_i)$, that can produce values at the output of this gate which do not appear in $T_{fv}(p_i)$. So we must restrict the inputs of this gate in several ways—multiple choices exist because it is a variant net—such that these disallowed values cannot appear at its output. When making these input restrictions, care must be taken to see that every input pattern from $T_{fv}(p_i)$ that yields permissible output values is accounted for in at least one of the choices. If the constraints imposed by a particular choice cannot be met, then another choice is selected. If there is no input combination that converts all variant nets into invariant ones, there exists no test pattern that sensitizes path p_i. Different heuristics can be used to decide the priority among different choices. For the sake of efficiency we would like to minimize both the number of choices and the overlap between different choices.

Example 1 Assume that the entries corresponding to a two-input AND gate (with input nets X_1 and X_2, output net Z) in $T_{fv}(p_i)$ is as follows:

X_1	X_2	Z
$0/\Delta/\overline{\Delta}$	$0/1/\Delta/\overline{\Delta}$	0

Note that net Z is variant since its value, as implied by nets X_1 and X_2, is $0/\Delta/\overline{\Delta}$. All the permissible input combinations that convert net Z into an invariant one are covered by the following three input patterns:

	X_1	X_2	Z
(i)	$\mathbf{0}$	$\mathbf{0/1/\Delta/\overline{\Delta}}$	$\mathbf{0}$
(ii)	$\mathbf{\Delta}$	$\mathbf{0/\overline{\Delta}}$	$\mathbf{0}$
(iii)	$\mathbf{\overline{\Delta}}$	$\mathbf{0/\Delta}$	$\mathbf{0}$

Depending on the heuristics used these three choices would have different priorities which in turn would decide the order in which they would be tried.

■

We now give a procedure that can be used to obtain the different choices that can be made to convert a variant net to an invariant one. Only a 2-input AND gate and a 2-input XOR gate need to be considered because, as stated earlier, all other cases can be derived from these. Consider a 2-input gate G with input nets X_1 and X_2 and output net Z. Let S_1, S_2, and S_G be the set of values in $\boldsymbol{T}_{fv}(\boldsymbol{p}_i)$, associated with X_1, X_2, and Z, respectively. Since Z is a variant net, $|S_1| > 1$ and $|S_2| > 1$. Without loss of generality, assume that $|S_2| \geq |S_1|$ and let $S_1 = \{s_{11}, s_{12}, \ldots, s_{1m}\}$. With S_G as the requested output value and $s_{1i} \in S_1$ as the value of one input we use either Table 8 (if G is an AND gate) or Table 3 (if G is an XOR gate) to obtain the set S_{2i}. The allowable value at input X_2, with s_{1i} at input X_1, is given by $S'_{2i} = S_{2i} \cap S_2$. This procedure is performed for all the elements of S_1. This yields the following choices:

Input X_1	$\{s_{11}\}$	$\{s_{12}\}$	$\ldots$	$\{s_{1m}\}$
Input X_2	S'_{21}	S'_{22}	$\ldots$	S'_{2m}

The output Z is an invariant net for any of the above choices because input X_1 has a single value. We may reduce the number of choices by combining values of input X_1. This can be done only when $S'_{2i} = S'_{2j}$. In this situation the i^{th} and j^{th} choices can be replaced by the input pattern that has $\{s_{1i}, s_{1j}\}$ at X_1 and S'_{2i} at X_2. The same procedure can be used to combine three choices if possible.

Example 2 Consider a 2-input AND gate (input nets X_1, X_2; output net Z) whose net entries in $\boldsymbol{T}_{fv}(\boldsymbol{p}_i)$ is shown below:

X_1	X_2	Z
$\mathbf{0/\Delta/\overline{\Delta}}$	$\mathbf{0/1/\Delta/\overline{\Delta}}$	$\mathbf{0/\Delta}$

.

Thus net Z is a variant net. The procedure described above yields the following choices:

Input X_1	$\mathbf{0}$	$\mathbf{\Delta}$	$\mathbf{\overline{\Delta}}$
Input X_2	$\mathbf{0/1/\Delta/\overline{\Delta}}$	$\mathbf{0/1/\Delta/\overline{\Delta}}$	$\mathbf{0/\Delta}$

Since the value of X_2 is identical for the first two choices they can be combined to yield:

Input X_1	$\mathbf{0}/\Delta$	$\overline{\Delta}$
Input X_2	$\mathbf{0}/\mathbf{1}/\Delta/\overline{\Delta}$	$\mathbf{0}/\Delta$

■

2.6 Use of Token Vectors

The introduction of $\Delta/\overline{\Delta}$ as a sensitized value was motivated by the deterministic propagation along an XOR/XNOR gate. For example, a Δ at the input of a two input XOR gate can produce both a Δ or a $\overline{\Delta}$ at the output depending on whether the other input is **0** or **1**. Thus unlike the other gates, XOR/XNOR gates may result in a situation where the parity of inversions with respect to the fault site can no longer be determined. In this section we discuss, how we can tackle this problem so that our deterministic implications are not weakened by this phenomenon.

Consider a gate G (not an XOR/XNOR gate) which lies on the path $\boldsymbol{p_i}$ that we deliberately sensitize. Evidently one input of G, say net m_ℓ, lies on path $\boldsymbol{p_i}$ and must have a sensitized value. If this value is Δ (or $\overline{\Delta}$) then our deterministic implication procedure would eliminate the value $\overline{\Delta}$ (or Δ) from the set of values of the other inputs of G. Consider the situation where net m_ℓ has the value $\Delta/\overline{\Delta}$ and the value of another input, say net m_k, of G contains both Δ and $\overline{\Delta}$. Also let the Δ and $\overline{\Delta}$ at nets m_ℓ and m_k be due to the value $\Delta/\overline{\Delta}$ at some FOS net m_j that influences both m_ℓ and m_k. Furthermore assume that a Δ ($\overline{\Delta}$) at net m_ℓ requires a Δ ($\overline{\Delta}$) at net m_j and that a Δ ($\overline{\Delta}$) at net m_k requires a $\overline{\Delta}$ (Δ) at net m_j. Thus, if we want to sensitize net m_ℓ, then net m_k cannot be sensitized. However since the value of net m_j contains both Δ and $\overline{\Delta}$ we would not be able to arrive at this conclusion using the implication rules alone. This motivates the introduction of the concept of "sensitization parity" which will help us in identifying such relationships among the sensitized values of different nets. For ease of explanation we introduce the following definitions:

Definition 1 Net m_j is said to be the **sensitization source** for net m_ℓ with respect to the fault site n if and only if all paths from net n to net m_ℓ pass through net m_j.

■

Note that the above definition does not necessarily imply that $m_\ell \in D(n)$ or that $m_\ell \in D(m_j)$.

Definition 2 The **path parity** of a single path p_α, not containing any XOR/XNOR gates, is the parity of the number of inverting gates along p_α.

■

We will use the value 0 for even parity and the value 1 to denote odd parity. As far as XOR (or XNOR) gates are concerned, the count of inversions is dependent on the exact inputs and not just circuit structure. Thus the path parity cannot be uniquely determined by circuit structure alone. The concept of path parity was effectively used in [2] for fault simulation purposes.

Definition 3 The **inversion parity** of net m_ℓ with respect to net m_j is b if and only if there exists at least one single path from net m_j to net m_ℓ and the path parity of all single paths from net m_j to net m_ℓ is b.

■

Definition 4 The **sensitization parity** of net m_ℓ with respect to net m_j, given a particular fault site n, is b if and only if net m_j is a sensitization source for net m_ℓ (with respect to the fault site n) and the inversion parity of net m_ℓ with respect to net m_j is b.

■

Let us consider again the gate G (not an XOR/XNOR gate) which lies on the path p_i that we deliberately sensitize. As before let the value of the input net m_ℓ of gate G that lies on path p_i be $\Delta/\overline{\Delta}$ and the value of another input, say net m_k, contain both Δ and $\overline{\Delta}$. If the sensitization parity of net m_ℓ with respect to net m_j is b and that of m_k with respect to net m_j is $\overline{b}$ then net m_k cannot be sensitized when we are trying to sensitize path p_i. Hence we can eliminate both Δ and $\overline{\Delta}$ from the value of net m_k. Note that we have excluded G to be an XOR/XNOR gate because the output of such a gate is sensitized if and only if it has an odd number of sensitized inputs which may include both Δ and $\overline{\Delta}$.

In order to take advantage of the information provided by the sensitization parity we introduce the concept of a token vector of the form $[m, b]$. If the token vector of net m_h is $[m_j, b]$, then b is the sensitization parity of net m_h with respect to net m_j. To explain the assignment of token vectors we divide gates (which are not XOR/XNOR gates) which have at least one input with a token vector and whose output token vector has not been assigned into two categories:

(i) Type I: gates for which all inputs with the TRUE token have token vectors.

(ii) Type II: gates for which there is at least one input with a TRUE token but no token vector.

In our procedure the token vector of the output net of a gate will be defined in terms of the input token vectors only when all the inputs with the TRUE token have identical token vectors. Otherwise we will restart the sensitization parity count at the output net. Thus the rule to assign token vectors is as follows:

If all the input token vectors of a gate G are identical (say, $[m, b]$), then its output is assigned $[m, b]$ if G is noninverting or $[m, \overline{b}]$ if G is inverting. Otherwise the output net m_g of gate G is assigned the vector $[m_g, 0]$.

In the algorithm for assigning token vectors we first level-order the net numbers of the circuit if it is not already in that format. A level ordered net numbering scheme implies that for every gate in the circuit the net number of the output is greater than the net number of any of its inputs.

We now describe the steps involved in assigning token vectors:

Step 1: Consider the set of all XOR/XNOR gates that have a TRUE token at their output net. Using the dominator forest inspect if any of these output nets influence a FOS net. If no such FOS net found then token vectors need not be assigned. If at least one such FOS net is found then continue.

Step 2: Assign $[m_s, 0]$ to the output of every XOR/XNOR gate that has a TRUE token at its output net m_s.

Step 3: If a net which was assigned a token vector in the previously executed step is an FOS net, then all its FOB nets are assigned the same vector.

Step 4: If there exists a gate of Type I, then assign its output token vector and go to Step 3. Otherwise, continue.

Step 5: If there exists a gate of Type II, then choose the one with lowest output net number and assign its token vector and then go to Step 3. If no such gates exist then the assignment of token vectors is complete.

Note that when the above procedure terminates (termination is due to the finiteness of the number of gates), all the gates (not XOR/XNOR gates) whose output nets do not have an assigned token vector are those for which no input has an assigned token vector.

There are two ways in which token vectors can provide information which a deterministic implication alone may not. If several inputs of a gate G (not an XOR/XNOR gate) have identical token vectors, then we may simultaneously sensitize any number of these inputs. Furthermore, we can never simultaneously sensitize two inputs of G whose token vectors differ only in their second component.

We will show in Appendix C that our algorithm for assigning token vectors satisfies the following properties:

Property 1 If the proposed algorithm assigns a token vector to a net then there exists a

path from the fault site to this net that contains an XOR/XNOR gate. ■

Property 2 If the proposed algorithm assigns a token vector to the output of a Type II gate G, then there is no XOR/XNOR gate in any path from the fault site to any input of G that has a TRUE token but no token vector. ■

Property 3 If the sensitization parity of net m_ℓ with respect to net m_j is b_1 and the algorithm assigns the token vector $[m, b]$ to net m_j then it would assign the token vector $[m, b \oplus b_1]$ to net m_ℓ. ■

Not all the token vectors generated by the above procedure will be useful—however their computation was necessary in order to compute the useful token vectors. The token vector of a net m_1 may be useful only if it is the input to a gate G (where G is not an XOR/XNOR gate) which has at least one more input, say net m_2, such that the first component of the token vectors of m_1 and m_2 are identical. Accordingly the token vector of any net that does not satisfy the above condition can be deleted.

Example 3 To illustrate the use of the token vectors consider the circuit shown in Figure 4. The token vectors of the nets, given that net 3 has a TRUE token, are shown in the figure. Assume the path being sensitized is through nets 3, 8, 12 and 14 and let the value of net 3 be $\Delta/\overline{\Delta}$. If token vectors were not used then the deterministic test cube that takes the propagation requirements into account would be as follows:

3	4	5	6	7	8	9
$\Delta/\overline{\Delta}$	0/1	0	$\Delta/\overline{\Delta}$	$\Delta/\overline{\Delta}$	$\Delta/\overline{\Delta}$	0/1

10	11	12	13	14
$1/\Delta/\overline{\Delta}$	$1/\Delta/\overline{\Delta}$	$\Delta/\overline{\Delta}$	$1/\Delta/\overline{\Delta}$	$\Delta/\overline{\Delta}$

In the above test cube nets are 13 and 14 are variant nets. If now the token vectors are take into account, then comparing those of nets 12 and 13 indicate that net 12 and 13 can never be simultaneously sensitized because their vectors differ only in their second component. Consequently net 13 must be set to **1**. The resulting deterministic test cube is:

3	4	5	6	7	8	9	10	11	12	13	14
$\Delta/\overline{\Delta}$	0	0	$\Delta/\overline{\Delta}$	$\Delta/\overline{\Delta}$	$\Delta/\overline{\Delta}$	0	1	1	$\Delta/\overline{\Delta}$	1	$\Delta/\overline{\Delta}$

Note that substantial deterministic changes result because of the use of token vectors and also that all the nets shown are now invariant.

■

3 Speed-up Techniques

3.1 Use of the Contrapositive

In this section we will discuss how using the contrapositive assertions of implications performed during the Pre-processing Phase can be used as an effective speed-up technique. The use of the contrapositive to reduce the search space was first suggested by Schulz, et. al., in SOCRATES [22].

The contrapositive of the logic expression $P \Longrightarrow Q$ is the equivalent expression $\sim Q \Longrightarrow \sim P$. Referring to the circuit of Figure 5 we notice that $X_3 = \mathbf{0} \Longrightarrow Z = \mathbf{0}$. Hence the contrapositive would yield $Z = \mathbf{1} \Longrightarrow X_3 = \mathbf{1}$. However, if we require the value **1** at Z given that all other nets have the value $\mathbf{0/1}$, no deterministic change would be implied by the backward implication procedure alone. Note, however, that in some cases a backward implication will yield the information provided by the contrapositive property. For example, $X_3 = \mathbf{0} \Longrightarrow Y_4 = \mathbf{0}$ yields $Y_4 = \mathbf{1} \Longrightarrow X_3 = \mathbf{1}$. However, a backward implication of $Y_4 = \mathbf{1}$ yields a **1** at X_3, X_4, and X_5. Hence it is useful to identify the conditions under which a backward implication cannot yield the information provided by a contrapositive assertion. In such cases we may store this information for possible use later in the test generation process.

The procedure presented in SOCRATES can only be used to backward imply the value 0 or 1 because it is assumed in [22, §4.3] that the "injected target fault is not located in this part of the circuit and the effects of the target fault cannot propagate to it as well." Furthermore, as mentioned in the *learning procedure* of [22, Fig. 5], the 0 and 1 implications are performed for all the nets of the circuit.

In this section we present a procedure that performs the **0** and **1** implications for only the FOS nets of the circuit. However we will show that this is sufficient to generate the information that can be obtained by performing the **0** and **1** implications for the remaining nets because of the deterministic nature of our backward implication procedure. Consequently the number of implications performed and the assertions stored for possible future use are less than that of SOCRATES. Furthermore we will also show that these stored **0** and **1** implications from FOS nets are sufficient to generate the useful contrapositive assertions for all 16 values of our system and for all nets of the circuit. Thus our use of contrapositive assertions will not be limited, as SOCRATES is, to only nets that are unaffected by the fault.

In our 16-valued system, assume that the forward implication of a value L_1 at net m_1 with $\mathbf{0/1}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$ at all other nets yields the value L_2 at net m_2. Thus when we require a value $L_2' \subseteq((\mathbf{0/1}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}})-L_2)$ at net m_2, then the value of net m_1 cannot contain any element

Value applied at net m_1	Implied value at net m_2								
	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	$(viii)$	(ix)
0	0/1	0	1	0	1	0	1	0/1	0/1
1	0/1	0	1	1	0	0/1	0/1	0	1

Table 4: Implications in **3-VP**

of L_1. To obtain the implications for all possible values of L_1 we only need to perform implications for each individual element of $\mathbf{0/1/\Delta/\overline{\Delta}}$. Thus the procedure to obtain the implications for the 16-valued system, henceforth referred to as **16-VP**, would be to set the value of net m_1 to each of the values **0**, **1**, Δ and $\overline{\Delta}$, one at a time and with $\mathbf{0/1/\Delta/\overline{\Delta}}$ at all other nets, and observe the implied value at net m_2. Each such implication is referred to as a **16-VP** "experiment." We will show that the information yielded by **16-VP** can be obtained from a simpler procedure that utilizes a 3-valued (**0**, **1**, **0/1**) logic system. In this procedure, which we denote as **3-VP**, we set the value of an FOS net m_1 to each of the values **0** and **1** one at a time and with **0/1** at all other nets, and observe the implied value at net m_2. Each such implication is referred to as a **3-VP** "experiment." For ease of explanation we define the values **0** and **1** as "singleton" values. Table 4 shows the nine possible combinations of values obtained by **3-VP** at net m_2 when the values **0** and **1** are applied at net m_1. Note that cases (ii) and (iii) show that net m_2 has a constant value independent of the circuit inputs. As a consequence, at least one of the stuck-at faults at net m_2 is undetectable. Cases (iv) and (v) simulate an identity function and an inverter between nets m_1 and m_2, respectively.

Consider a **3-VP** experiment performed by setting a FOS net m_1 to a singleton value and all other nets to **0/1**. The experiment in which net m_1 is set to the same singleton value while all other nets are set to $\mathbf{0/1/\Delta/\overline{\Delta}}$ is known as the corresponding **16-VP** experiment. The following theorems, whose proofs appear in Appendix D, establish the relationship between the results of a **3-VP** experiment and the corresponding **16-VP** experiment.

Theorem 1 If a **3-VP** experiment yields a singleton value at net m_2, then the corresponding **16-VP** experiment yields the same singleton value at this net.

■

Theorem 2 If a **3-VP** experiment yields the value **0/1** at net m_2, then the corresponding **16-VP** experiment yields the value $\mathbf{0/1/\Delta/\overline{\Delta}}$ at this net.

■

Value applied at net m_1	Implied value at net m_2								
	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	$(viii)$	(ix)
0	$0/1/\Delta/\overline{\Delta}$	0	1	0	1	0	1	$0/1/\Delta/\overline{\Delta}$	$0/1/\Delta/\overline{\Delta}$
1	$0/1/\Delta/\overline{\Delta}$	0	1	1	0	$0/1/\Delta/\overline{\Delta}$	$0/1/\Delta/\overline{\Delta}$	0	1

Table 5: Implications of a **0** and **1** in **16-VP**

	Value applied at net m_1	Implied value at net m_2								
		(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	$(viii)$	(ix)
3-VP	0	0/1	0	1	0	1	0	1	0/1	0/1
	1	0/1	0	1	1	0	0/1	0/1	0	1
16-VP	0	$0/1/\Delta/\overline{\Delta}$	0	1	0	1	0	1	$0/1/\Delta/\overline{\Delta}$	$0/1/\Delta/\overline{\Delta}$
	1	$0/1/\Delta/\overline{\Delta}$	0	1	1	0	$0/1/\Delta/\overline{\Delta}$	$0/1/\Delta/\overline{\Delta}$	0	1
	Δ	$0/1/\Delta/\overline{\Delta}$	0	1	Δ	$\overline{\Delta}$	$0/\Delta$	$1/\overline{\Delta}$	$0/\overline{\Delta}$	$1/\Delta$
	$\overline{\Delta}$	$0/1/\Delta/\overline{\Delta}$	0	1	$\overline{\Delta}$	Δ	$0/\overline{\Delta}$	$1/\Delta$	$0/\Delta$	$1/\overline{\Delta}$

Table 6: Relationship between **3-VP** and **16-VP**

Consequently Table 5 is obtained from Table 4 when a **0** and **1** implication is performed in **16-VP**. We now show that the information yielded by **16-VP** can be obtained from that yielded by **3-VP**. We do this by illustrating how Table 5 can be used to obtain the implications due to a $\boldsymbol{\Delta}$ or a $\overline{\boldsymbol{\Delta}}$ at net m_1. Note that a $\boldsymbol{\Delta}$ at net m_1 corresponds to a change in value of net m_1 from a 1 to a 0 or a 0 to a 1. Thus, to obtain the implied value at net m_2 due to a $\boldsymbol{\Delta}$ at net m_1 we only need to know the value at net m_2 due to a **1** and a **0** at net m_1 in **16-VP**.

Consider the situation where a **1** and a **0** at net m_1 yields a **0** and $\mathbf{0/1/\Delta/\overline{\Delta}}$, respectively, at net m_2. Thus, with a **0** at net m_1 we can obtain both **0** and **1** at net m_2. Thus if the (b_g, b_f) value at net m_1 is (1,0) then the possible (b_g, b_f) values at net m_2 are (0,0) and (0,1). On the other hand a (b_g, b_f) of (0,1) at net m_1 implies the possible values at net m_2 are (0,0) and (1,0). This information can be represented in a compact form by stating that the value $\boldsymbol{\Delta}$ ($\overline{\boldsymbol{\Delta}}$) at net m_1 implies the value $\mathbf{0/\overline{\Delta}}$ ($\mathbf{0/\Delta}$) at net m_2. Analogously, we can inspect the other cases to generate the implications of a $\boldsymbol{\Delta}$ or $\overline{\boldsymbol{\Delta}}$ at net m_1. Table 6 summarizes our results and shows how the **16-VP** table can be obtained from the **3-VP** table.

We now discuss the conditions under which the information yielded by a contrapositive assertion cannot be obtained by a deterministic backward implication alone and hence should be stored for future use. Consider the situation where a singleton value L_1 at net m_1 yields, using **3-VP**, a singleton value L_2 at the output net m_2 of a gate G. The corresponding contrapositive assertion should be stored if and only if the value L_2 can be obtained at the

L_2	G			
0	OR	NAND	XOR	XNOR
1	NOR	AND	XOR	XNOR

Table 7: (L_2, G) combinations that yield useful contrapositive assertions

output of G by setting all its inputs to non-controlling values. Consequently, Table 7 shows the L_2 and G combinations for which this implication should be stored for future use. In general, for the cases that satisfy the (L_2, G) combinations given in Table 7 we will not be able to drop L_1 from the set of all possible values at net m_1 when we require a value $L_2' \subseteq((\mathbf{0/1/\Delta/\overline{\Delta}})-L_2)$ at net m_2 by using only the backward implication procedure.

We now present a procedure which, when incorporated into the Pre-processing Phase, can derive all the contrapositive assertions for our 16-valued system.

1. Construct two test cubes $\boldsymbol{tc}_{oo}$ and $\boldsymbol{tc}_{o1}$ in which the values of all nets of the circuit are set to $\mathbf{0/1}$.

2. In $\boldsymbol{tc}_{oo}$ ($\boldsymbol{tc}_{o1}$) change the value of net m_1, where m_1 is a FOS net , to the singleton value $L_1(\overline{L}_1)$ and perform a forward implication of this value.

   Let L_2 (L_3) be the implied value at the output net m_2 of gate G.

3. If both L_2 and L_3 are singleton values, then both these implications (L_1 at $m_1 \Longrightarrow L_2$ at m_2 and $\overline{L}_1$ at $m_1 \Longrightarrow L_3$ at m_2) need to be stored.

4. If only one of the values (say L_2) is singleton and this value L_2 and the gate G happen to be one of the combinations listed in Table 7, then this implication (L_1 at $m_1 \Longrightarrow L_2$ at m_2) should be stored. Note that whenever we have only one implication stored for a given m_1, m_2 pair it means that the other value at net m_1 yields a $\mathbf{0/1}$ at net m_2.

5. Repeat steps 1–4 for all FOS nets.

The *learning procedure* presented in SOCRATES [22, Fig. 5] performs the 0 and 1 implications for all nets of the circuit. However, we have reduced the amount of computation and storage requirements by performing the implications for only FOS nets. It is easy to show that the information for all other nets can be derived from this because of the deterministic nature of our backward implication procedure.

The contrapositive assertions in the 16-valued system corresponding to the implications stored by the above procedure can be generated using Table 6. So, it has to be shown that if any implication was not stored by the above procedure, then either its corresponding

contrapositive assertion yields no information or the information yielded can be derived by using the stored contrapositive assertions and the backward implication rules. The former situation refers to the trivial case when both L_2 and L_3 are **0/1**. The latter situation refers to the case where only one of L_2 or L_3 (say L_2) is singleton and (L_2, G) is not one of the combinations listed in Table 7. This implies that a controlling value at some input of G produces the value L_2 at its output. Hence there exists at least one input of G, say net m_j, such that the value of L_1 at net m_1 yields, using **3-VP**, a controlling value for gate G at net m_j. From Lemma 3 of Appendix D we know that this **3-VP** experiment creates a path (say $\boldsymbol{p}_1$) of singleton values from net m_1 to net m_j. Furthermore, if the singleton value of any net on this path is a non-controlling value for the gate G_i it drives, then this experiment sets all inputs of G_i to non-controlling values. Let us consider the two possible ways this could happen:

(i) in path $\boldsymbol{p}_1$ all nets have values that are controlling values for the gates they drive.

(ii) In path $\boldsymbol{p}_1$ there exists at least one gate such that the value L_1 at net m_1 sets all inputs of such gates to non-controlling values. Let G_i be the gate which satisfies the above property and is closest to m_2. Thus, using Table 7, the implication of the value, say L_i, obtained at the output of G_i due to the value L_1 at net m_1 would be stored for future use.

Consider the situation where the required value at net m_2 does not include the value L_2. Thus we can drop the controlling input value from all inputs of G, including net m_j. Furthermore if the value at net m_2 does not include $\boldsymbol{\Delta}$ ($\overline{\boldsymbol{\Delta}}$) then we can drop the value $\boldsymbol{\Delta}$ ($\overline{\boldsymbol{\Delta}}$) from all inputs of G, including net m_j, if G is a non-inverting (or inverting) gate. We then traverse backwards along path $\boldsymbol{p}_1$ from net m_j to net m_1. In the case of situation (i) described above we would be able to drop, from every net on path $\boldsymbol{p}_1$, the controlling value for the gate this net drives. Moreover, we could also drop one $\boldsymbol{\Delta}$ or $\overline{\boldsymbol{\Delta}}$ (depending on the number of inversions along $\boldsymbol{p}_1$) from these nets too. Thus at net m_1 we will be able, by applying the backward implication procedure, to drop the value L_1 and one of the values $\boldsymbol{\Delta}$ or $\overline{\boldsymbol{\Delta}}$. If situation ($ii$) had occurred then we would be able to drop the controlling value and one of $\boldsymbol{\Delta}$ or $\overline{\boldsymbol{\Delta}}$ from nets on path $\boldsymbol{p}_1$ till we reach the output of gate G_i. Specifically we would drop the value L_i and one of the values $\boldsymbol{\Delta}$ or $\overline{\boldsymbol{\Delta}}$ (as is appropriate) from the set of values at the output of gate G_i. However since we have stored the implication (L_1 at net $m_1 \Longrightarrow L_i$ at output of G_i) we would now invoke the contrapositive of this assertion to drop L_1 and the appropriate $\boldsymbol{\Delta}$ or $\overline{\boldsymbol{\Delta}}$ from the set of values of net m_1. Thus the implications stored by the procedure outlined in this section are sufficient to generate all the necessary contrapositive assertions.

The discussion in this section was so far limited to performing forward implications from nets and storing only those whose contrapositive assertions are useful. However, as pointed

out in [16], performing the backward implication can also yield useful relationships between the values of the nets. As an example consider the circuit shown in Figure 6.

Performing backward implication of the value **0** at net S yields the relation (**0** at net $S \Longrightarrow$ **1** at net R) from which we can deduce that (**0** at net $R \Longrightarrow$ **1** at net S). Thus if we had stored this implication it could potentially be useful during the test generation process. Note that obtaining all the deterministic changes due to a backward implication of a value at some net will generally involve performing forward implication of the values of nets also. This is unlike the case for forward implication where we did not have to perform backward implication of the values of nets to obtain the useful contrapositive assertions. For example we need to forward imply the value **0** at P and the value **1** at Q in order to obtain the value **1** at R due to a **0** at S. The procedure proposed in [16] performs backward implication (which may involve performing forward implications too) from all nets of the circuit, analyzes the results of these implications and accordingly stores those which may be useful later, i.e. those which cannot be directly obtained from forward or backward implication. Moreover, the procedure in [16] can only be applied to nets that are unaffected by the fault site. As in the case of forward implications, the relationship between **3-VP** and **16-VP** for backward implications can also be derived from Table 6. This is easily understood by realizing that a backward implication, using **3-VP**, of a singleton value at the output of a gate given a **0/1** at all inputs can only result in changes at the inputs of the gate if the requested output is due to all inputs being set to non-controlling values. Thus both **3-VP** and **16-VP** would give the same results in such a situation. If some input of the gate has already been set to a singleton value **3-VP** then it must be due to a forward implication and hence **16-VP** would also yield the same result. Thus as in the case of forward implications, we can also use the contrapositive of backward implications for nets that affected by the fault being considered. But, in order to obtain <u>all</u> useful contrapositive assertions, we need to perform implications for <u>all</u> nets of the circuit. A method to obtain a subset of all useful contrapositive assertions by performing implications for only some of the nets of the circuit is proposed in [5].

3.2 Identifying Independent Subcircuits During Enumeration Phase

Recall that in the Enumeration Phase we have to convert all variant nets to invariant ones with values that are subsets of their required values in the deterministic test cube being considered. In this section we discuss how we can use the dominator forest to identify "independent" subcircuits whose value justification during the Enumeration Phase can be done independently.

The concept of postponing the value justification of nets whose justification process is guaranteed to succeed was proposed in FAN [11]. It was based on classifying all the nets of

the circuit into the following categories:

- When a signal line L is reachable from some fan-out point, that is, there exists a path from some fan-out point to L, we say that L is a *bound line.*
- A signal line which is not bound is said to be a *free line.*
- When a free line is adjacent to some bound line, we say that it is a *head line.*

In [11] the backtrace procedure terminates at a head line instead of continuing towards a PI. This is because a head line is the output of a fanout-free sub-circuit and hence its value justification can always be satisfied. Consequently it is useful to postpone the value justification of head lines till all other lines have been justified. This avoids a lot of unnecessary computation in the event that one of the other lines cannot be justified. Unfortunately, in most practical circuits, including the ISCAS benchmark circuits [7], a majority of the head lines are the PI nets. This is because almost all PI nets are also FOS nets. Thus significant improvement in ATPG performance cannot be expected to be contributed by using the concept of head lines.

To overcome this difficulty, TOPS [17] introduced the concept of "basis nodes" whereby a net (say m) is defined to be a basis node if and only if all FOS nets that influence m totally reconverge prior to it. Utilizing this property TOPS could postpone the value justification of basis nodes until that of other nodes because they do not interfere with the value justification of nets lying outside its cone of influence. Furthermore, if the circuit does not contain any nets whose value is constant (i.e. independent of the PIs) then the value justification of the basis nodes will not lead to contradictions. Although the use of basis nodes is a generalization of the head line concept introduced in FAN [11], it is still a static procedure and does not take into account the constraints imposed by the values of the test cube generated at any stage of the test generation. Moreover both the head line and basis node concepts are limited to the the discussion of nets which are unaffected by the fault site. In this section we present a dynamic procedure that overcomes these drawbacks by utilizing both the circuit structure and the values of all the nets in the associated deterministic test cube. Furthermore our procedure can be applied to all nets of the circuit, including those that are affected by the fault.

We will denote a variant net as a **Satisfiable Variant Net** (SVN) if the justification of its value with respect to a deterministic test cube is guaranteed to succeed and thus can be postponed to the last stage of test generation.

In some cases it may be possible to identify nets which are not necessarily SVNs but their value justification depends on a subset of the PIs which do not influence the value

justification of some other variant nets. In such a situation the value justification of the two sets of nets in question are independent. Thus for a given deterministic test cube it would be useful to identify these nets – henceforth denoted as **Independent Variant Nets** (IVNs) – so that their value justification can proceed independently. Note that with every IVN there is an associated subcircuit such that the value justification of the IVN is independent of all nets outside this subcircuit. Hence it is important to identify this subcircuit along with the IVN.

The procedure for the identification of SVNs and IVNs should be substantially simpler than their actual value justification for this speed-up technique to be useful. Thus we will concentrate on developing fairly straightforward procedures that utilize information that has been already derived by other aspects of the test generation process and try to identify as many SVNs and IVNs as possible.

For the remainder of this section we will refer to a net as being "single-valued" if the cardinality of the set of values associated with this net, in the deterministic test cube being considered, is one. Similarly a net will be termed "multi-valued" if the cardinality is greater than one.

We now present procedures for the identification of SVNs and IVNs using the dominator forest and the values of the circuit nets in the deterministic test cube with respect to which the nets in question are variant.

3.2.1 SVN Identification

Let m_v be the net under inspection which is variant with the value L_v in the deterministic test cube $\boldsymbol{d(tc)}$. Consider the subtree $\mathcal{T}_{m_v}$ of the dominator forest that has net m_v as its root.

(*i*) From $\mathcal{T}_{m_v}$ delete all nodes that correspond to FOS nets. Note that the removal of a node implies the removal of the entire subtree which has this node as its root. The remaining tree corresponds to the largest fanout-free subcircuit that has net m_v as its output.

(*ii*) Consider all the nodes of the remaining tree which are single-valued in $\boldsymbol{d(tc)}$. Remove every child of each of these nodes. Let us denote the remaining tree as $\mathcal{T}'_{m_v}$.

Consider all the leaves of $\mathcal{T}'_{m_v}$ which corresponds to nets which are multi-valued in $\boldsymbol{d(tc)}$. If all these nets are PIs then net m_v is an SVN. In other words, if net m_v is an SVN then the inputs of the fanout-free subcircuit corresponding to $\mathcal{T}'_{m_v}$ can be divided into two categories — nets which are single-valued in $\boldsymbol{d(tc)}$ and PI nets with the value $\mathbf{0/1}$ in $\boldsymbol{d(tc)}$. As an example of the SVN identification procedure consider the subcircuit shown in Figure 7 which is part of a larger circuit. Note that the value **1** at PI net 3 is obtained by using the contrapositive assertions. In this example net 12 is a variant net and inspection of $\mathcal{T}'_{12}$ shows

that it is indeed an SVN.

Let m_v be a net which has been verified to be an SVN with respect to some $\boldsymbol{d(tc)}$ as per the identification procedure described. Consider the subcircuit corresponding to the tree $\mathcal{T}'_{m_v}$ that is constructed by the procedure. The general structure of this fanout-free subcircuit is depicted in Figure 8 where the leaves of $\mathcal{T}'_{m_v}$ consist of single-valued (all values discussed are with respect to $\boldsymbol{d(tc)}$) nets $m_{s1}, m_{s2}, \ldots, m_{sk}$ and PI nets $m_{i1}, m_{i2}, \ldots, m_{ij}$. Note that some or all of $m_{s1}, m_{s2}, \ldots, m_{sk}$ may also be variant in $\boldsymbol{d(tc)}$. Consider the set of nodes consisting of all the interior nodes (all nodes except the leaves) of $\mathcal{T}'_{m_v}$ except the root net m_v. All nodes belonging to this set are multi-valued and some of them may also be variant. Let L_v be the required value, in $\boldsymbol{d(tc)}$, at the SVN net m_v. The following theorem provides the rationale for postponing the value justification of an SVN until all the non-SVN nets have been converted to invariant ones.

Theorem 3 For every $l_v \in L_v$, there exists an assignment of the PI nets $m_{i1}, m_{i2}, \ldots, m_{ij}$ that converts net m_v to an invariant net with the value l_v. Furthermore this assignment will also convert all the other interior variant nets of $\mathcal{T}'_{m_v}$ into invariant ones with values that are subsets of their required values in $\boldsymbol{d(tc)}$.

Proof Since $\mathcal{T}'_{m_v}$ corresponds to a fanout-free circuit and all values are with respect to a deterministic test cube $\boldsymbol{d(tc)}$, thus for every $l_v \in L_v$ there exists an assignment of the unassigned inputs of $\mathcal{T}'_{m_v}$ (in this case the PI nets $m_{i1}, m_{i2}, \ldots, m_{ij}$) that will result in the value l_v at net m_v thereby converting it to an invariant net. Furthermore this assignment converts all the nets of $\mathcal{T}'_{m_v}$ into single-valued nets. Consider the situation where one of the previously variant interior nets, say m_i, of $\mathcal{T}'_{m_v}$ now has a value that is not one of the required values at this net. Thus this value is capable of producing the value l_v at net m_v. Moreover because of the fanout-free structure of $\mathcal{T}'_{m_v}$, net m_i affects the rest of the circuit through net m_v. Hence the value obtained at net m_i due to this assignment would not be dropped from it's set of allowable values because it produces a required value at net m_v. Hence the obtained value at m_i must be one of the elements of its required value in $\boldsymbol{d(tc)}$. Thus all the interior nets of $\mathcal{T}'_{m_v}$ are also converted to invariant ones. ■

Thus for every single-value $l_v \in L_v$, there exists an assignment of these PIs which yield the value l_v at net m_v without interfering with the values of nets outside $\mathcal{T}'_{m_v}$. The following theorem, whose proof follows directly from the SVN identification procedure, is useful in the process of justifying SVNs.

Theorem 4 Let m_v be an SVN with respect to some deterministic test cube $\boldsymbol{d(tc_1)}$. Let $\boldsymbol{d(tc_2)}$ be a deterministic test cube obtained by converting some variant nets in $\boldsymbol{d(tc_1)}$ to invariant ones by assigning some of the unassigned PIs of $\boldsymbol{d(tc_1)}$. Then net m_v is either an SVN or an invariant net with respect to $\boldsymbol{d(tc_2)}$. ■

3.2.2 IVN Identification

As in the case of SVNs, in order to check whether a variant net m_v is an IVN with respect to a deterministic test cube we start with the subtree $\mathcal{T}_{m_v}$ of the dominator forest that has net m_v as its root.

(*i*) For every node m_s of $\mathcal{T}_{m_v}$ which is single-valued in the deterministic test cube in question, consider the subtree $\mathcal{T}_{m_s}$ which has net m_s as its root.

(*ii*) If none of the FOB leaves of $\mathcal{T}_{m_s}$ are multi-valued then delete the subtree $\mathcal{T}_{m_s}$ from $\mathcal{T}_{m_v}$. Otherwise, consider the FOS nets corresponding to the multi-valued FOB leaves of $\mathcal{T}_{m_s}$. If all these FOS nets are outside $\mathcal{T}_{m_v}$ then delete the subtree $\mathcal{T}_{m_s}$ from $\mathcal{T}_{m_v}$. After all possible deletions let the remaining subtree of $\mathcal{T}_{m_v}$ be denoted as $\mathcal{T}''_{m_v}$.

(*iii*) If for every multi-valued FOB leaf of $\mathcal{T}''_{m_v}$, the corresponding FOS net also belongs to $\mathcal{T}''_{m_v}$ then net m_v is an IVN.

The value justification of net m_v can be performed by assigning values to the multi-valued PI leaves of $\mathcal{T}''_{m_v}$ and is independent of all nets that are not in $\mathcal{T}''_{m_v}$. However unlike the situation for SVNs, this value justification process is not guaranteed to succeed. It is important to note that net m_v need not be the only variant net in $\mathcal{T}''_{m_v}$. In such a situation the value justification of all the variant nets in $\mathcal{T}''_{m_v}$ are dependent — however it is independent of all nets lying outside $\mathcal{T}''_{m_v}$.

As an example of a situation where a net can be ascertained to be a IVN and yet its value in a certain $\boldsymbol{d(tc)}$ cannot be justified, consider the circuit of Figure 9. The output of the XOR gate is a IVN with respect to a $\boldsymbol{d(tc)}$ which has the values shown in the figure. However, enumeration will show that this net can only have the value **1** given the values present at the FOB nets shown. The important thing to realize, however, is that even though the value justification of a IVN may not succeed, the justification process is independent of the rest of the circuit.

We now discuss the rationale for the proposed procedure for IVN identification. Note that a basis node is an IVN with respect to any deterministic test cube. This is because a node m_v is a basis node if and only if all the FOS nets corresponding to the FOB leaves of $\mathcal{T}_{m_v}$ are contained in $\mathcal{T}_{m_v}$. In order to potentially identify more nets whose value justification is independent of other nets we can then relax this condition to allow the single-valued FOB leaves of $\mathcal{T}_{m_v}$ to have their FOS nets outside $\mathcal{T}_{m_v}$. This is because the value of these nets will not be changed during the value justification of net m_v and will not affect any nets outside $\mathcal{T}_{m_v}$. More IVNs can potentially be identified if we delete single-valued nodes from $\mathcal{T}_{m_v}$ provided the value justification of net m_v does not result in an incorrect value at the single-valued node m_s that was deleted. The value justification of net m_v can affect the value of node m_s by changing the value of the FOB leaves of the tree $\mathcal{T}_{m_s}$. Inspection of the

multi-valued FOB leaves of $\mathcal{T}_{m_s}$ can result in one of the following three situations:

(i) All the FOS nets corresponding to the multi-valued FOB leaves of $\mathcal{T}_{m_s}$ belong to $\mathcal{T}_{m_v}$.

(ii) All the FOS nets corresponding to the multi-valued FOB leaves of $\mathcal{T}_{m_s}$ are outside $\mathcal{T}_{m_v}$.

(iii) There is at least one multi-valued FOB leaf of $\mathcal{T}_{m_s}$ whose FOS net is outside $\mathcal{T}_{m_v}$ and at least one multi-valued FOB leaf of $\mathcal{T}_{m_s}$ whose FOS net is in $\mathcal{T}_{m_v}$.

Situation (i) does not violate the requirement of a basis node but the FOB leaves of $\mathcal{T}_{m_s}$ can be affected by the value justification of m_v and hence $\mathcal{T}_{m_s}$ should not be deleted from $\mathcal{T}_{m_v}$. In situation (ii) the value justification of net m_v will not affect the value of net m_s provided net m_v is an IVN as per the procedure described earlier. Hence $\mathcal{T}_{m_s}$ can be deleted from $\mathcal{T}_{m_v}$. We now explain why we cannot delete the subtree $\mathcal{T}_{m_s}$ from $\mathcal{T}_{m_v}$ when we have situation (iii). Consider the situation depicted in Figure 10 where nets m_1 and m_2 are the FOB nets corresponding to the FOS net m_{12} and nets m_3 and m_4 are the FOB nets corresponding to the FOS net m_{34}. Let m_v be the variant net being inspected and let m_s be a single-valued node in its tree. Furthermore let nets m_{12} and m_{34} be multi-valued in the deterministic test cube being considered. Note that the presence of m_1 and m_3 would prevent us from deleting $\mathcal{T}_{m_s}$ from $\mathcal{T}_{m_v}$. During the value justification of net m_v, net m_{12} might be set to a certain value which in turn will assign this new value to nets m_1 and m_2. This new value of net m_1 might impose certain conditions on the value of net m_3 in order that the required value of net m_s be satisfied. Consequently this will affect the value of net m_{34} and hence value justification of net m_v will no longer be independent of nets lying outside $\mathcal{T}_{m_v}$.

Note that a SVN is also an IVN — however the distinction between SVNs and IVNs is of importance because we can prioritize the value justification of variant nets. In this strategy nets which are neither IVNs nor SVNs will be justified before IVNs which in turn will be justified before SVNs. Thus if any stage results in a contradiction then the subsequent stages need not be performed.

3.3 Backward Implication of Desensitizing Values

In this subsection we discuss how backward implication of the desensitizing value from variant nets may help speed-up the enumeration process. Consider the output net m_1 of a gate G_1 which has the value L_1 and is a variant net in $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \boldsymbol{k}))$, a deterministic test cube obtained at some stage of the Enumeration Phase. Let L_1' be the value implied at net m_1 by the values in $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \boldsymbol{k}))$ of the inputs of G_1. We construct a new test cube $\boldsymbol{d}'(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \boldsymbol{k}))$ which is identical to $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \boldsymbol{k}))$ except that net m_1 has the value $L_1' - L_1$. Note that the value $L_1' - L_1$ at net m_1 desensitizes path $\boldsymbol{p}_i$. Using $\boldsymbol{d}'(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \boldsymbol{k}))$ we backward imply

the value $L_1' - L_1$ at net m_1 by applying only the backward implication rules and the stored contrapositive assertions and observe the nets whose values change in the process. Let m_j, $2 \leq j \leq J$, be the nets where this backward implication terminates. Note that m_j is either a PI or the output of a gate whose input values do not change during this process. Also, let $L_j', 2 \leq j \leq J$, be the new value obtained at net m_j by the above procedure.

Since the value $L_1' - L_1$ at net m_1 implies that the value of net m_j is L_j', we know from the contrapositive principle that, for *any* j, $2 \leq j \leq J$, if the value of net m_j does not contain any of the values in the set L_j', then the value at net m_1 will not contain any of the values in the set $L_1' - L_1$ and hence m_1 will become an invariant net. A sufficient condition to make m_1 an invariant net without interfering with the requirements of other variant nets is that there exists some m_j such that $m_1 \in D(m_j)$ and m_j is a basis node. If m_j is not a basis node but is a SVN in the new deterministic test cube formed after removing the value L_j' form its value in $\boldsymbol{d}(\boldsymbol{tc}_{fv}(\boldsymbol{p}_i, \boldsymbol{k}))$, then net m_1 can be made invariant.

3.4 Selection of Alternate Sensitizing Paths

Suppose that it is not possible to generate a test from $\boldsymbol{T}_{fv}(\boldsymbol{p}_i)$. This means that there exists no test for the given *s-a-v* fault that sensitizes path $\boldsymbol{p}_i$. We now have to select another path, if any exists (say p_j), and construct the corresponding $\boldsymbol{T}_{fv}(\boldsymbol{p}_j)$, if possible, to generate a test for the fault. The following theorem may be helpful in deciding whether there exists a test for the fault that sensitizes path $\boldsymbol{p}_j$.

Theorem 5 If the value of every net in $\boldsymbol{T}_{fv}(\boldsymbol{p}_j)$ is a subset of the corresponding value in $\boldsymbol{T}_{fv}(\boldsymbol{p}_i)$ (denoted by $\boldsymbol{T}_{fv}(\boldsymbol{p}_j) \subseteq \boldsymbol{T}_{fv}(\boldsymbol{p}_i)$) and there is no input pattern that sensitizes path $\boldsymbol{p}_i$, then there is no input pattern that sensitizes path $\boldsymbol{p}_j$.

Proof. (By contradiction.) Assume that there is an input pattern $\boldsymbol{I}$ that sensitizes path $\boldsymbol{p}_j$. Since $\boldsymbol{T}_{fv}(\boldsymbol{p}_j)$ imposes only those restrictions that must be satisfied by any input pattern that sensitizes path $\boldsymbol{p}_j$, then values of all the nets of the circuit in the presence of input $\boldsymbol{I}$ must be subsets of their corresponding values in $\boldsymbol{T}_{fv}(\boldsymbol{p}_j)$. Since $\boldsymbol{T}_{fv}(\boldsymbol{p}_j) \subseteq \boldsymbol{T}_{fv}(\boldsymbol{p}_i)$, then the values of all the circuit nets in the presence of input $\boldsymbol{I}$ must be subsets of their corresponding values in $\boldsymbol{T}_{fv}(\boldsymbol{p}_i)$. Thus $\boldsymbol{I}$ satisfies all the restrictions imposed by $\boldsymbol{T}_{fv}(\boldsymbol{p}_i)$ and hence it sensitizes path $\boldsymbol{p}_i$, which is a contradiction. ■

3.5 Examples

Example 4 Let us reconsider the circuit of Figure 2 with net 25 as the fault site to highlight the improvements obtained by the speed-up techniques proposed in this section.

As before we will use test cubes to reflect the results of each stage of the test generation. Also the entries shown for a particular test cube include only those nets whose values are different from those in the preceding cube. Note that the initial test cube tc_1 will be identical to that obtained earlier. However $d(tc_1)$ is different because use of the contrapositive principle yields further changes. The value of net 47 and the use of the contrapositive of the assertion (**1** at net 42 $\Longrightarrow$ **1** at net 47) drops the value **1** from net 42 which in turn changes the value of nets 39, 40, 8 and 12. Similarly the value of net 41 and use of the contrapositive principle changes the value of nets 4, 19 and 20 to **1**. The resulting $d(tc_1)$ is:

4	8	12	19	20	26	27	28	33	34	35	39	40
1	**0**	**0**	**1**	**1**	Δ	Δ	Δ	$0/\Delta$	$0/\overline{\Delta}$	$0/\Delta$	$0/\Delta$	$0/\overline{\Delta}$

41	42	43	44	47	49	50	51	52
0	$0/\Delta/\overline{\Delta}$	$0/\Delta/\overline{\Delta}$	$0/\Delta/\overline{\Delta}$	$0/\Delta/\overline{\Delta}$	$\Delta/\overline{\Delta}$	$\Delta/\overline{\Delta}$	$0/\Delta/\overline{\Delta}$	$1/\Delta/\overline{\Delta}$

As before we extend the sensitized path through net 28 and obtain tc_2 in which the value of net 35 is changed to Δ. From tc_2 construct $d(tc_2)$, which is shown below:

11	13	29	30	34	36	40	42	43	44
1	**1**	**1**	**1**	**0**	**1**	**0**	$0/\Delta$	$0/\Delta$	$0/\Delta$

45	46	47	48	49	50	51	52
$0/1/\Delta$	$0/1/\Delta$	$0/\Delta$	Δ	Δ	Δ	$0/\Delta$	**1**

Note that the resulting $d(tc_2)$ contains more information than the corresponding $d(tc_2)$ shown earlier because of the use of contrapositive assertions. From (**0** at net 42 $\Longrightarrow$ **0/1** at net 47) and (**1** at net 42 $\Longrightarrow$ **1** at net 47) we can use Table 6 to conclude that ($\overline{\Delta}$ at net 42 $\Longrightarrow$ $1/\overline{\Delta}$ at net 47). Thus the value $0/\Delta$ at net 47 drops the value $\overline{\Delta}$ at net 42 which in turn sets the value of nets 40, 34 and 11 to **0**, **0** and **1** respectively. The $0/\Delta$ at net 42 also changes the value of nets 43, 44, 45 and 46. We then extend the sensitized path through net 49 to obtain the tc_3 and $d(tc_3)$ shown below.

tc_3:

51
Δ

$d(tc_3)$:

1	17	18
1	**1**	**1**

At this point we have a sensitized path from the fault site to a PO and must construct $T_{f_1}(p_i)$ and $T_{f_0}(p_i)$ in order to generate tests for both stuck-at faults. As before we restrict our attention only to the s-a-0 fault at net 25 and construct the $T_{f_0}(p_i)$ shown below by setting net 25 to **1** and finding all its deterministic implications.
$T_{f_0}(p_i)$:

9	10	21	22	23	24
1	**1**	**1**	**1**	**1**	**1**

We now invoke the Enumeration Phase to convert the variant nets (nets 41 and 47 in this case) into invariant ones. We first prioritize the variant nets so that the value justification of SVNs can be postponed till the other variant nets have been taken care of. In order to do this we first identify the subtrees T'_{41} and T'_{47}, as defined in §3.2, for nets 41 and 47 respectively. Using the procedure described, we can conclude that net 41 is an SVN and net 47 is not – thus we first try to justify the value of net 47. We can now perform the backward implication of the desensitizing value **1** at net 47 to obtain the value **1** at nets 15 and 16 as shown in Figure 11. Thus setting either of the PI nets 15 or 16 to the value **0** will yield the required value at net 47 and would convert it to an invariant net. We can now proceed to justify the value of net 41 – a process we which we know is guaranteed to succeed. For example, setting the PI nets 2 and 3 to a **1** is a possible solution.

Once again we note that the generation of tests for the s-a-1 faults at the checkpoint nets 22 and 23, while sensitizing the same path p_1, involves only the construction of $T_{f_1}(p_1)$ with appropriate values for nets 22 and 23 and then executing the Enumeration Phase. In summary, we have used the same $T_f(p_1)$, which is $d(tc_3)$ of our example, to obtain tests for four single stuck-at faults. These are the faults net 22 s-a-0, net 22 s-a-1, net 23 s-a-0 and net 23 s-a-1.

■

Example 5 Consider the class of circuits shown in Figure 12 with net 3 being the fault site. Note that the ECAT circuit considered by Goel to illustrate the efficiency of PODEM [14] is an element of this class. Using $D(3) = \{5, 7\}$ we construct tc_1 as shown below where all other nets have the value **0/1**.

1	2	3	3_f	4	5	6	7
0/1	**0/1**	**0/1**	Δ	**0/1**	$\Delta/\overline{\Delta}$	**0/1**	$\Delta/\overline{\Delta}$

The attempt to construct $d(tc_1)$ does not change the value of any net i.e. tc_1 is a deterministic test cube. Moreover we have a sensitized path from 3_f to the PO and hence

we can invoke the Enumeration Phase after setting the value of net 3 depending on the type of stuck-at fault being tested. For example, if we have to generate a test for net 3 s-a-0 then we set the value of net 3 to **1** and construct the corresponding deterministic test cube $T_{f_0}(p_i)$. This causes the value of nets 1 and 2 to change to **1**. Since we have a sensitized path from 3_f to the PO and there are no variant nets, a test has been generated. Note that the algorithm specifies only the value of PI nets 1 and 2 because it takes full advantage of the linearity of XOR gates.

■

4 Identification of Redundant Faults

The existence of redundant faults in circuits is one of the principal reasons that cause test generation to be such a computationally intensive exercise. Most ATPG schemes are complete algorithms in the sense that given enough time they will implicitly try all input combinations before declaring a target fault to be redundant. The ability to quickly identify redundancies is a good measure of the efficiency of test generation algorithms. As pointed out earlier, SIMPLE tries to achieve this goal by considering all the necessary assignments that arise out of sensitizing a particular path from the fault site to a PO. In spite of this, the fact remains that this kind of redundancy identification is a by-product of actual test generation. In this section we investigate several other approaches that can enhance the performance of SIMPLE by identifying more undetectable faults without actually testing for them. Some of these approaches will be based on using information generated during the Pre-Processing Phase to identify possible undetectable faults, while others will use the knowledge of already identified redundant faults to determine newer ones. However before discussing these redundancy identification techniques let us review some information that lends a proper perspective to the material presented in this section.

The need to identify redundancies does not arise in circuits that are either fanout free or do not possess reconvergent fanouts because a circuit can be redundant with respect to some single or multiple stuck-at fault only if it contains reconvergent fanout. It is also well known that if the fault x_i s-a-v is undetectable, where x_i is a primary input and $v \in \{0,1\}$, then the fault x_i s-a-$\overline{v}$ is also undetectable.

All ATPG algorithms use concepts like fault equivalence and fault dominance in order to reduce the set of faults in a circuit that need to be tested. One approach is to consider only single stuck-at faults on the *checkpoints* of a circuit where the checkpoints comprise the PIs and FOBs of the circuit [6]. This is because a test set that detects all single stuck-at faults on the checkpoints of a circuit C (that contains NOT, AND, NAND, OR and NOR gates

only) detects all single stuck-at faults in C [6]. Thus the initial list of *target faults* can consist of the checkpoint faults. However as pointed out in [1, 10, 19] a test set that detects all detectable checkpoint faults does not necessarily detect all detectable single stuck-at faults in the circuit. Thus if any of the checkpoint faults is undetectable then additional target faults may need to be considered. It has been shown in [1] that for any set of target faults based on dominance relations, the only faults not guaranteed to be detected by a test set that detects all detectable target faults are those faults which dominate only undetectable faults. Using this result, the authors of [1] provided a method for identifying additional target faults.

Definition 5 [1] The term *direct checkpoint* of a line l denotes a checkpoint k such that there exists a unique path between k and l and no other checkpoint lies on this path.

■

Theorem 6 [1] Let T be a test set that detects all detectable checkpoint faults and let l *s-a-v* be a detectable fault not detected by T. Every direct checkpoint of l has an undetectable single stuck-at fault.

■

Since SIMPLE allows the use of XOR and XNOR gates as primitive elements we need to extend the concept of checkpoints to *generalized checkpoints* which will consist of all the PI nets, FOB nets and the output nets of all XOR and XNOR gates of the circuit. This is because unlike the situation for other gates, a test set that detects all stuck-at faults at the inputs of an XOR/XNOR gate does not necessarily detect the stuck-at faults at the output. Consequently we will also extend the definition of a direct checkpoint to that of a *generalized direct checkpoint* in order to include the output of XOR/XNOR gates. We now discuss ways in which knowledge of an undetectable fault can be used to identify more redundant faults.

4.1 Redundancy Identification using Known Undetectable Faults

In this section we will discuss the properties of a circuit that allow us to identify undetectable faults by using the information about already determined undetectable faults.

Theorem 7 Let net m be a basis node in a circuit C such that net m does not implement a constant (i.e. 0 or 1). If the fault m *s-a-v* is undetectable, then the fault m *s-a-*$\overline{v}$ is also undetectable. Furthermore the stuck-at faults at all the nets in the subcircuit which has net m as its output are also undetectable.

Proof Let $\vec{z}$ be the function implemented by C. Since h is a basis node we can partition the inputs $\vec{x} = (x_1, x_2, \ldots, x_n)$ into two disjoint components $\vec{x_a}$ and $\vec{x_b}$ where $h = H(\vec{x_a})$ and $\vec{z} = F(h, \vec{x_b})$. Since h *s-a-v* is undetectable, and h does not implement a constant value, $dF/dh = 0$ for all possible values of $\vec{x_b}$ and independent of the value of $\vec{x_a}$, where dF/dh is the Boolean difference of F with respect to h. Hence h *s-a-*$\overline{v}$ will also be undetectable as will be the stuck-at faults at all nets which have net h as its dominator and must propagate sensitization through it. ■

The above theorem suggests that we try to generate tests for basis nodes instead of first trying to generate tests for its direct checkpoints. If no test can be generated for the faults at a basis node then the faults at its direct checkpoints (actually the entire subcircuit that drives this basis node) are automatically determined to be redundant. On the other hand, this method is still useful if the faults at the basis node are detectable. In fact, as soon as we determine the test for any stuck-at fault at a basis node, the test generation for the other stuck-type fault and all faults in its subcircuit involve enumeration only because the propagation requirements beyond the basis node have already been accounted for.

In order to analyze how topological dominance relates to redundant faults we introduce the following definition.

Definition 6 Net m_2 is said to be an even (odd) dominator of net m_1 if and only if net m_2 is a dominator of net m_1 and all paths from m_1 to m_2 have even (odd) parity.

■

Note that we cannot determine the parity of a path which contains an XOR or XNOR gate.

Theorem 8 If net m_2 is an even (odd) dominator of net m_1 and the fault net m_2 *s-a-v* is undetectable then the fault net m_1 *s-a-v* ($\overline{v}$) is also undetectable.

Proof (By contradiction for the even dominator case — the odd dominator case is analogous). Let the test t detect the fault net m_1 *s-a-v*. Then the (b_g, b_f) value of net m_1 due to t must be $(\overline{v}, v)$. Since m_2 is an even dominator of m_1, the (b_g, b_f) value of net m_2 in the presence of t must also be $(\overline{v}, v)$. Thus the fault m_2 *s-a-v* is also activated by t. Since t propagates the effect of the fault m_1 *s-a-v* beyond net m_2 to some PO and m_2 is a dominator of m_1, t also tests the fault m_2 *s-a-v*. This is a contradiction — hence the fault net m_1 *s-a-v* must be undetectable. ■

Corollary 1 If a circuit is redundant with respect to a single stuck-at fault α at net m_1 then it is redundant with respect to some single stuck-at fault β at net m_2, where m_2 is a generalized direct checkpoint of net m_1. ■

In [5] we have presented several properties of a circuit that allow us to identify undetectable faults from the knowledge of known redundant PI faults.

4.2 Redundancy Identification using Topological Information

In this section we will discuss how certain redundant faults can be identified by using the information about the circuit topology in conjunction with some of the experiments performed during the Pre-Processing Phase of SIMPLE.

Definition 7 Let G_ℓ be a gate whose output is net m_ℓ. An input m_i of G_ℓ is defined to be an *off dominator sensitizing input* of net m if and only if net m_i does not lie on any path from net m to any PO and net m_ℓ belongs to the set of dominators, $D(m)$, of net m.

■

For any gate G let $I_{NC}(G)$ and $I_C(G)$ denote the non-controlling and controlling input value of G respectively. Similarly let $O_{NC}(G)$ denote the value that is obtained at the output of G if all its inputs are set to non-controlling values and $O_C(G)$ denote the value obtained at the output if at least one input is set to a controlling value. The next two theorems, whose proofs are straightforward, illustrates the usefulness of the off dominator sensitizing inputs of a net in detecting undetectable faults.

Theorem 9 Let m_{sj} be a FOB net corresponding to a FOS net m_s. Let m_i be an off dominator sensitizing input of net m_{sj} and let this net m_i drive the gate G_i. If during **3-VP**, applying the value v at net m_s yields the value $I_C(G_i)$ at net m_i, then the fault net m_{sj} s-a-$\overline{v}$ is undetectable.

■

Consider the situation during **3-VP** where a value v at a FOS net m_s yields the value $O_C(G_i)$ at the output of a gate G_i. If this happens then inspect each input m_i of G_i to see whether it is a FOB net of m_s or a dominator of a FOB net of m_s. If this is true and if at least one of the remaining inputs of G_i (i.e. inputs other than m_i) has the value $I_C(G_i)$ in this **3-VP** experiment then the FOB net in question is redundant with respect to the s-a-$\overline{v}$ fault.

As an example of this situation consider the circuit shown in Figure 13 which is a subcircuit of c6288, one of the ISCAS benchmark circuits [7]. The implication of a 0 at FOS net 591, during **3-VP**, yields a 0 at net 1371, which is the output net of a gate whose input net 593 is a FOB of net 591. Furthermore the other input of this gate (net 1313) has a controlling input value in this experiment. Thus the fault net 593 s-a-1 is undetectable. Similarly, since

a 1 at net 591 yields a 0 at net 1401, net 1371 is a dominator of net 593, and net 1372 has a controlling value, we can deduce that the fault net 593 s-a-0 is also undetectable. In fact all the faults in c6288 which have been reported by [4] to be undetectable (both stuck-at faults at nets 593, 641, 689, 737 and 785) can be easily verified to be so by the application of Theorem 9 i.e. during the Pre-Processing Phase.

Theorem 10 Let the FOB net m_{i1} be an off dominator sensitizing input of net m_j. Let net m_{si} be the FOS net corresponding to net m_{i1} and let G_{i1} be the gate driven by m_{i1}. Let net m_{i2} be another off dominator sensitizing input of net m_j and let the gate driven by net m_{i2} be denoted as G_{i2}. If during **3-VP**, applying the value $I_{NC}(G_{i1})$ at net m_{si} yields the value $I_C(G_{i2})$ at net m_{i2}, then both the stuck-at faults at net m_j are undetectable.

■

As an example of a situation where the above theorem can be applied, consider the circuit shown in Figure 14. In this figure, nets b and d are off dominator sensitizing inputs of net a. In **3-VP**, performing the implication of the value **1** (which is the non-controlling input value for gate G_1) at net c yields the value **1** at net d. This value is a controlling value for gate G_2 and hence both stuck-at faults at net a are undetectable.

5 Conclusion

In this report we have presented a 16-valued ATPG algorithm that introduces several new concepts to make test generation more efficient. It is the only algorithm that takes into account all the deterministic implications of sensitizing a path prior to the enumeration process. The resulting ability to identify inconsistencies prior to enumeration improves the possibility of quicker identification of redundant faults. Instead of sensitizing a single gate at a time, we sensitize subpaths by sensitizing all gates lying between successive FOS nets, thereby reducing the number of times deterministic test cubes have to be constructed. Our algorithm exploits the linearity of the XOR/XNOR gate and also shows how use of the sensitization parity can speed-up test generation for circuits containing such gates. The speed-up techniques introduced are based on both circuit topology and the net values that must be satisfied to sensitize the chosen path. We have also shown how use of the desensitizing values can guide the selection of PIs in the Enumeration Phase. The contrapositive assertions for our 16-valued system was obtained from a simple procedure that uses a 3-valued system and performs forward implications for FOS nets only. We emphasize that the different aspects of the algorithm discussed above owe their efficiency to the strength of the 16-valued system

used. We have also shown how the dominator forest of a circuit can be effectively used in several phases of test generation.

We have also shown how to exploit the common requirements that are imposed when we sensitize the same path from the fault site to a PO in order to generate tests, whenever possible, for both stuck-at faults at this net.

Various properties that can be used to identify redundant faults without actual test generation have been presented. This is an important fact because redundant faults seem to be the major bottleneck in any ATPG algorithm.

Note that there are no changes involved in the procedure described if we are interested in performing conventional stuck-at fault testing at a net as opposed to fault site testing. In such a situation we only need to interpret Δ and $\overline{\Delta}$ as D and $\overline{D}$ respectively and use the initializing values discussed earlier.

Appendix

A Construction of Deterministic Test Cubes

In a $\boldsymbol{d}(\boldsymbol{tc}_k)$ all deterministic implications (no arbitrary choice) of all entries of the test cube $\boldsymbol{tc}_k$ are fully considered. For example, if the output of an AND gate is $\mathbf{0/1/\Delta/\overline{\Delta}}$ and one of its inputs changes to $\mathbf{0/\Delta}$, then, irrespective of the other inputs, the output is changed to $\mathbf{0/\Delta}$.

To construct $\boldsymbol{d}(\boldsymbol{tc}_1)$ from $\boldsymbol{tc}_1$ we perform backward and forward implications of all nets whose values in $\boldsymbol{tc}_1$ are different from $\mathbf{0/1}$ and $\mathbf{0/1/\Delta/\overline{\Delta}}$ and all other nets whose values change during this implication process. In the general case, when we are constructing $\boldsymbol{d}(\boldsymbol{tc}_k)$ from $\boldsymbol{tc}_k$, we start by considering the forward and backward implications of the nets whose values in $\boldsymbol{tc}_k$ are different from those in the last successfully constructed deterministic test cube and that of all other nets whose values change during this implication process. During the construction of $\boldsymbol{d}(\boldsymbol{tc}_k)$ from $\boldsymbol{tc}_k$, if a backward or forward implication request results in a new value L_j' for any net m_j of the circuit, then we should update the corresponding net entry L_j by setting it to $L_j \cap L_j'$. If this intersection yields the empty set then $\boldsymbol{d}(\boldsymbol{tc}_k)$ cannot be constructed.

In order to obtain $\boldsymbol{d}(\boldsymbol{tc}_k)$ the process of forward and backward implications should be continued until no more changes occur in the values associated with any net. Note that this process will terminate in a finite number of steps because we are performing set intersection on finite sets.

The rules for constructing deterministic test cubes must include the provision for appropriately handling the values of nets associated with fanout points and should also take into account the information provided by the token vectors. We now present the rules for forward and backward implication.

A.1 Forward Implication

The process of forward implication of the values associated with every net is done with the help of Tables 1, 2 and 3. These tables are a generalization of the truth tables of the respective gates. For gates with more than two inputs the method adopted is similar to that used by Akers [3]. We view every gate as being constructed out of 2 input gates and use the existing values at the inputs of a gate to generate a new value for the output. Depending on the gate in question, appropriate tables are used. Note that the three tables are sufficient because OR, NOR, and NAND functions can be derived by appropriately using Tables 1

and 2, whereas Tables 2 and 3 can be used to generate the XNOR function. Also we do not have to perform forward implication for gate G_f — its output value is determined by the initialization described earlier.

Suppose we are performing forward implications due to change(s) in input(s) of a gate G whose output is net m. Let L_O be the set of values associated with net m in the test cube prior to forward implication being performed. Also let L_N be the value obtained at net m by using the new values of the inputs of G. Net m will then be set to $L_O \cap L_N$ unless $L_O \cap L_N = \emptyset$ which implies a contradiction. Four other situations are possible:

1. $L_O = L_N$. No further action is needed for this forward implication.

2. $L_N \subset L_O$ (proper subset). We now have to consider the forward implication of the value of L_N at net m on all gates driven by G.

3. $L_O \subset L_N$. We now have to perform a backward implication of the value L_O at net m. This may result in further changes in the inputs of gate G.

   Example 6 An example of the situation where $L_O \subset L_N$ is shown in Figure 15. If input A of gate G is changed from **0/1** to **1**, then forward implication using Table 1 would yield $L_N =$ **0/1**. Since $L_O \subset L_N$, we now perform a backward implication of the value **0** at the output of gate G. It will be clear from the next section that this backward implication yields a **0** at input B.

   ■

4. $L_O \not\subseteq L_N$ and $L_N \not\subseteq L_O$. Both forward and backward implications of the value $L_O \cap L_N$ at net m should be performed.

   Example 7 An example of the situation where $L_O \not\subseteq L_N$ and $L_N \not\subseteq L_O$ can be seen from the incompletely specified circuit of Figure 16. Assume that at some stage of test generation we have the following $\boldsymbol{d(tc_k)}$.

1	2	3	4	5	6	7	8
0/1	**Δ**	**Δ**	**Δ**	**0/1**	**0/1**	**0/1**	**1/Δ**

9	10	11	12	13	14	15	16
0/Δ	**0/1**	**0/Δ**	**0/Δ**	**0/1/Δ**	**0/Δ**	**0/Δ/$\overline{\Delta}$**	**Δ/$\overline{\Delta}$**

* / **	0	1	Δ	$\overline{\Delta}$
0	$0/1/\Delta/\overline{\Delta}$	$\emptyset$	$\emptyset$	$\emptyset$
1	0	1	Δ	$\overline{\Delta}$
Δ	$0/\overline{\Delta}$	$\emptyset$	$1/\Delta$	$\emptyset$
$\overline{\Delta}$	$0/\Delta$	$\emptyset$	$\emptyset$	$1/\overline{\Delta}$

* Requested Output

** Existing value at one input

Table 8: Backward Implication for a 2-input AND gate

The value at net 14 is $L_O = \mathbf{0}/\Delta$. If we now extend the sensitized path through net 4 by setting nets 5, 6 and 7 to **1** then forward implication would yield the value $L_N = \mathbf{1}/\Delta$ at net 14. Hence $L_O \cap L_N = \Delta$, $L_O \not\subseteq L_N$ and $L_N \not\subseteq L_O$.

■

A.2 Backward Implication

The process of backward implication involves determining the changes required at the inputs of a gate in order to satisfy a requested change at the output. A change in the value of a net will mean that one or more possible values associated with the net has been deleted. In that sense an input change can be made only if the deleted value can never be used with the existing values at the other inputs to generate any of the requested output value(s).

Example 8 Consider a two-input AND gate whose values at inputs and output is $\mathbf{0}/\Delta$. If we require that the output be changed to **0**, we cannot change any of the inputs because all the input values can be used in some input pattern to generate a **0** at the output.

■

A general set of backward implication rules can be derived in terms of the input values and the requested output value. However, in a manner similar to that presented in [3] we consider each multiple input gate as a cascade of two input gates. The backward implication rules for a two-input AND gate is shown in Table 8. Note that the element $\emptyset$ has been included in this table to detect an unsatisfiable backward implication request. The complete

table for all 15 non-$\emptyset$ values is obtained by the set union operation. If we set $\boldsymbol{\Delta}$= $\boldsymbol{D}$ (or $\overline{\boldsymbol{\Delta}}$= $\boldsymbol{D}$) then the resulting table is equivalent to that proposed by Akers [3]. To perform backward implication for a two-input AND gate we reference the table using the requested value at the output and the existing value at one input to generate the value of the other input. Since the XOR gate is linear, Table 3 can be used for backward implication also. Thus Tables 2, 3 and 8 can be used to perform backward implication for any two-input gate. Irrespective of the gate in question, the value generated by the appropriate table must be intersected with the existing value of the input to generate the new value of the input. Analogously, the new value of the input and the requested value of the output must now be used to generate the new value of the other input. For example, consider a 2-input gate whose input values are L_1 and L_2. If the requested value of the output of the gate is L_G, then we use L_G and L_1 to determine the new value L_2' of the second input and then L_2' and L_G to determine the new value L_1' of the first input.

Example 9 Consider the two-input AND shown in Figure 17. Initially, input A has the value $\mathbf{0}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$, input B has the value $\mathbf{0}/\mathbf{1}/\boldsymbol{\Delta}$, and the output has the value $\mathbf{0}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$. If we require a $\boldsymbol{\Delta}$ at the output, then the backward implication process using Table 8 and values of C and A would yield a $\mathbf{1}/\boldsymbol{\Delta}$ at B. That, intersected with its existing value of $\mathbf{0}/\mathbf{1}/\boldsymbol{\Delta}$, yields $\mathbf{1}/\boldsymbol{\Delta}$. Now a backward implication of a $\boldsymbol{\Delta}$ at C with a $\mathbf{1}/\boldsymbol{\Delta}$ at B yields $\mathbf{1}/\boldsymbol{\Delta}$ at A. This value of A intersected with the existing value of $\mathbf{0}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$ results in a $\boldsymbol{\Delta}$ at input A.

■

As stated before, any gate with more than two inputs will be represented as a cascade of two-input gates. Consider an n-input gate G represented as a cascade of $(n-1)$ two-input gates G_1, $G_2, \ldots, G_{n-2}$ and G_{n-1}, with net numbers as shown in Figure 18. Assume that the values at nets $1, 2, \ldots, n$ are $X_1, X_2, \ldots, X_n$ respectively. We first use forward implication of these values to compute $Y_1, Y_2, \ldots, Y_{n-2}$, the values of nets $n+1, n+2, \ldots, n+(n-2)$ respectively. Then using the value Z, which is the required value at the output of gate G, we apply the backward implication rules for gate G_{n-1} to obtain Z_{n-2} and X_n', the new values of nets $n+(n-2)$ and n respectively. Having done that, we proceed backwards and apply the backward implication rules for all the gates, one at a time, ending with gate G_1. Since the binary operation represented by any logic gate is associative, the order in which the inputs X_i are cascaded is irrelevant.

It will be shown in Appendix B that the above procedure will stabilize in a single pass, unlike the approach followed in [3] which may require several passes.

Example 10 Consider the 3-input XOR gate G shown in Figure 19 with associated net values. Assume that we request the value $\boldsymbol{\Delta}$ at net 5. We now view gate G as constructed

out of 2-input gates G_1 and G_2 as shown in Figure 19. The values of nets 1 and 2 are first used to compute the value $\mathbf{0/1}/\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$ at net 4. By using Table 3 and the requested value $\boldsymbol{\Delta}$ at the output of G_2 we obtain the value $\mathbf{0/1}$ at net 4, but the value of net 3 remains unchanged. By requesting a $\mathbf{0/1}$ at net 4 we obtain a $\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$ at input net 2, and the value of net 1 remains unchanged.

■

Note that in the above example the value of the intermediate net 4 does not have to be sensitized in order that the overall gate output be sensitized. This can only happen for XOR/XNOR gates.

From the discussion on backward implication it should be clear that it is not always possible to make changes at the inputs of a gate G such that the new value of the inputs yield exactly the requested value at the output of the gate (in the above example the new values of the inputs produce a $\boldsymbol{\Delta}/\overline{\boldsymbol{\Delta}}$ at the output of G). In Appendix B it is shown that the requested value L_G at the output is always a subset of L'_G, the value at the output implied by the new values of the inputs obtained by the backward implication procedure.

B Properties of the Backward Implication Procedure

In this appendix we discuss some useful properties of the backward implication procedure. For ease of explanation we will use the following notation in this appendix. Let G be a two-input gate. If A and B are the set of values of the inputs of G, then the set of values of the output, implied by these inputs, is denoted by $G(A, B)$. Also, let (L_0/L_i) denote the set of values at one input that can produce L_0 at the output of the gate, given that the other input is L_i.

Property 1. Let G be a two-input gate with inputs A and B. Consider a backward implication of the value Z, where $Z \subseteq G(A, B)$, at the output of G. If this backward implication causes the inputs to be changed to A' and B', respectively, then

$$Z \subseteq Z'$$

where $Z' = G(A', B')$.

Proof. Let $z \in Z$. Since $Z \subseteq G(A, B)$ there exists $a \in A$ and $b \in B$ such that $\{z\} = G(\{a\}, \{b\})$. Since $z \in Z$, after a backward implication of Z at the output of G is performed using the given tables, the new values of the inputs A' and B' are such that $a \in A'$ and $b \in B'$. Thus $z \in G(A', B')$, i.e., $z \in Z'$. Therefore $Z \subseteq Z'$. ■

Note that the above property can be extended to gates which have more than two inputs.

Consider a two-input gate whose input values are L_1 and L_2 and the requested value at the output is L_G. The new values of the inputs after one pass of the backward implication procedure will be

$$L_2' = L_2 \cap (L_G/L_1)$$

and

$$L_1' = L_1 \cap (L_G/L_2').$$

If $L_2' \subseteq (L_G/L_1')$ then another pass of the backward implication procedure is unnecessary because $L_2' \cap (L_G/L_1')$ is going to yield L_2'—implying no further changes at the input.

Property 2. $L_2' \subseteq (L_G/L_1')$

Proof. Select any $l_2 \in L_2'$. Thus $l_2 \in (L_G/L_1)$. Hence there exists $l_1 \in L_1$ such that $\{l_g\} = G(\{l_1\}, \{l_2\})$ where $l_g \in L_G$. Since $l_2 \in L_2'$ and $G(\{l_1\}, \{l_2\}) \subseteq L_G$ then $l_1 \in (L_G/L_2')$ and consequently $l_1 \in L_1'$. Therefore $l_2 \in (L_G/L_1')$ since $l_1 \in L_1'$ and $G(\{l_1\}, \{l_2\}) \subseteq L_G$. This proves that $L_2' \subseteq (L_G/L_1')$. ■

As a consequence of Property 2, every new application of the backward implication procedure requires only a single pass to determine the new values of each input of the gate in question.

C Proof of Properties of Token Vectors

In this appendix we will prove some of the properties of the algorithm, for assigning token vectors, that were presented in §2.6. The proof of **Property 1** is straightforward and will not be presented.

Proof of Property 2 By definition, a Type II gate G must have an input that has a TRUE token but no token vector. Consider any path from the fault site to such an input (say m_i) of G. If the path contains an XOR/XNOR gate it would have been assigned a token vector in Step 2. Consider every net on this path from the XOR/XNOR gate to net m_i. All these nets have a lower net number than the output of G and consequently would have been assigned a token vector before it. Hence net m_i must also have an assigned token vector. This is a contradiction. ■

Proof of Property 3 Since the sensitization parity of net m_ℓ with respect to net m_j is b_1 and net m_j has an assigned token vector, then all nets on any path from net m_j to net m_ℓ must have TRUE tokens and there are no XOR/XNOR gates on any of these paths. Thus the algorithm will assign a token vector to net m_ℓ. We now show by contradiction that the algorithm assigns the token vector $[m, b \oplus b_1]$ to net m_ℓ. Assume that the first component of the token vector assigned by the algorithm to net m_ℓ is different from m. Thus there

exists at least one gate G on a path from net m_j to net m_ℓ whose token vector at the output is $[m', x]$ where $m' \neq m$ and which has at least one input with a token vector whose first component is m. Trace this path starting from net m_j till the first such gate is encountered. This implies that there exists at least one input to this gate whose token vector is $[m, 0]$ and at least one input whose token vector is $[m, 1]$. Consequently we cannot define the inversion parity of the output net of this gate with respect to net m_j, which in turn implies that net m_j cannot be a sensitization source for net m_ℓ. This is a contradiction.

Thus the algorithm assigns m as the first component of the token vector of net m_ℓ. In fact, it assigns m as the first component of the token vector of all nets on every path from net m_j to net m_ℓ. The fact that the algorithm assigns $b \oplus b_1$ as the second component of the token vector of net m_ℓ is a consequence of the above fact and the definition of sensitization parity. ■

D Proof of Theorems in §3.1

In this appendix we will always denote the net at which we apply a value (either in **3-VP** or **16-VP**), to observe the implications at other nets, as net m_1. In order to prove the theorems stated in §3.1 we will make use of the following lemmas. The proofs of the first three lemmas are straightforward and will not be presented.

Lemma 1 The value of every net in a **3-VP** experiment is a subset of the value of this net in the corresponding **16-VP** experiment. ■

Lemma 2 In a **3-VP** experiment, if we traverse backwards along any path of singleton values from net m_j which has a singleton value, then we will always reach net m_1. ■

Lemma 3 If a **3-VP** experiment yields a singleton value at net m_j, then this experiment yields a path of singleton values from net m_1 to net m_j. Furthermore, if this experiment yields a singleton value at the output net of any gate G, then it either sets one or more of its inputs to controlling values or all its inputs to non-controlling values. ■

Lemma 4 If the output of a gate G has a singleton value in a **3-VP** experiment and additional value(s) in the corresponding **16-VP** experiment, then there exists at least one input to this gate which has a singleton value in this **3-VP** experiment and additional value(s) in this **16-VP** experiment.

Proof (i) Assume that in the **3-VP** experiment the singleton value at the output of G was obtained due to a controlling value at one or more inputs of G. In this situation all these

inputs must contain additional value(s) in the corresponding **16-VP** experiment. Otherwise, we will obtain the singleton value at the output as given by the **3-VP** experiment.

(*ii*) From Lemma 3, the only remaining alternative is that the singleton value at the output of gate G in the **3-VP** experiment was obtained by setting all its inputs to non-controlling values. Thus it is obvious that at least one input must contain additional value(s) in the corresponding **16-VP** experiment in order to produce additional value(s) at the output. ■

Proof of Theorem 1 Let G be the gate whose output is net m_2 or whose output is a FOS net with m_2 as one of its FOB nets. Let L_2 be the singleton value yielded at net m_2 by the **3-VP** experiment. Let the corresponding **16-VP** experiment yield the value L_2' at net m_2. By Lemma 1, $L_2 \subseteq L_2'$. Assume that $L_2 \subset L_2'$. Then, by Lemma 4, at least one input of G must contain a singleton value in this **3-VP** experiment and additional value(s) in this **16-VP** experiment. This input must either be net m_1 or be connected to the output of some gate. If the latter is true, then this gate must also possess at least one input that has a singleton value in this **3-VP** experiment and additional value(s) in this **16-VP** experiment. Proceeding backwards in this manner we must, by Lemma 2, eventually conclude that net m_1 has a singleton value in this **3-VP** experiment and additional value(s) in this **16-VP** experiment. This is a contradiction. Thus, $L_2 = L_2'$. ■

Proof of Theorem 2. Let G be the gate whose output is net m_2 or whose output is a FOS net with m_2 as one of its FOB nets. Since net m_2 has the value **0/1** in the **3-VP** experiment, then at least one input of G must also have the value **0/1** in this experiment and all the values of all other inputs must include the non-controlling value for gate G. The input with the **0/1** value is either a primary input or is connected to the output of some gate which in turn must have at least one input with the value **0/1** and the value of all other inputs must include the non-controlling value for this gate. Proceeding in this manner we will reach some primary input, say I_1, in a finite number of steps. Thus there is a path $\boldsymbol{p}_1$ from I_1 to net m_2 such that all nets on this path have the value **0/1** in this **3-VP** experiment. Furthermore, the value of all inputs of every gate on this path $\boldsymbol{p}_1$ must contain the non-controlling value. By Lemma 1, in the corresponding **16-VP** experiment, the value of all inputs of every gate along path $\boldsymbol{p}_1$ must contain the non-controlling value. Thus a $\mathbf{0/1/\Delta/\overline{\Delta}}$ from input I_1 will propagate through every gate along path $\boldsymbol{p}_1$ yielding a $\mathbf{0/1/\Delta/\overline{\Delta}}$ at net m_2. ■

References

[1] M. Abramovici, P. R. Menon and D. T. Miller, "Checkpoint Faults are not sufficient Faults for Test Generation," *IEEE Transactions on Computers*, Vol. C-35, pp 770–771, August 1986.

[2] M. Abramovici, P. R. Menon and D. T. Miller, "Critical Path Tracing - An Alternative To Fault Simulation," in *Proceedings of the* 20^{th} *ACM/IEEE Design Automation Conference*, pp 214–220, 1983.

[3] Sheldon B. Akers, "A Logic System for Fault Test Generation," Presented at Symposium on Fault-Tolerant Computing, Paris, France, June 1975. Also *IEEE Transactions on Computers*, Vol. C-25, pp 620–630, June 1976.

[4] Sheldon B. Akers, Christie Joseph and Balakrishnan Krishnamurthy, "On the Role of Independent Fault Sets in the Generation of Minimal Test Sets," *Proceedings of the IEEE International Test Conference*, pp 1100–1107, 1987.

[5] Akhtar-uz-zaman M. Ali, "Use of a 16 Valued Logic System in Combinational Circuit Testing," PhD Dissertation, Syracuse University, August 1990.

[6] M. A. Breuer and A. D. Friedman, *Diagnosis & Reliable Design of Digital Systems.* Computer Science Press, 1976.

[7] Frank Brglez and Hideo Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," in *Proceedings of the IEEE Symposium on Circuits and Systems; Special Session on ATPG and Fault Simulation,* pp 663–698, June 1985.

[8] Charles W. Cha, William E. Donath and Füsun Özgüner, "9-V Algorithm for Test Pattern Generation of Combinational Digital Circuits," *IEEE Transactions on Computers*, Vol. C-27, pp 193–209, March 1978.

[9] Wu-Teng Cheng, "Split Circuit Model for Test Generation," in *Proceedings of the* 25^{th} *ACM/IEEE Design Automation Conference,* pp 96–101, 1988.

[10] Warren Debany, "On Using the Fanout-Free Substructure of General Combinational Networks," PhD Dissertation, Syracuse University, December 1985.

[11] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Transactions on Computers*, Vol. C-32, pp 1137–1144, December 1983.

[12] H. Fujiwara and S. Toida, "The complexity of fault detection: An approach to design for testability," in *Proceedings of the* 12^{th} *International Symposium on Fault Tolerant Computing*, pp 101–108, June 1982.

[13] John Giraldi and Michael L. Bushnell, "EST: The New Frontier in Automatic Test-Pattern Generation," in *Proceedings of the* 27^{th} *ACM/IEEE Design Automation Conference,* pp 667–672, June 1990.

[14] Prabhakar Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, Vol. C-30, pp 215—222, March 1981.

[15] O. H. Ibarra and S. K. Sahni, "Polynomially Complete fault detection problems," *IEEE Transactions on Computers*, Vol. C-24, pp 242–259, March 1975.

[16] R. Jacoby, P. Moceyunas, H. Cho and G. Hachtel, "New ATPG Techniques for Logic Optimization," *Proceedings of the IEEE International Conference on Computer Aided Design,* pp 548–551, 1989.

[17] Tom Kirkland and M. Ray Mercer, "A Topological Search Algorithm for ATPG," in *Proceedings of the* 24^{th} *ACM/IEEE Design Automation Conference,* pp 502–508, 1987.

[18] P. Muth, "A Nine-Valued Circuit Model for Test Generation," *IEEE Transactions on Computers*, Vol. C-25, pp 630–636, June 1976.

[19] Y. W. Ng and A. Avizienis, "Comments on 'Fault Folding for Irredundant and Redundant Combinational Circuits'," *IEEE Transactions on Computers*, Vol. C-25, pp 207, February 1976.

[20] Janusz Rajski and Henry Cox, "A Method of Test Generation and Fault Diagnosis in Very Large Combinational Circuits," *Proceedings of the IEEE International Test Conference*, pp 932–943, September 1987.

[21] J. P. Roth, W. G. Bouricius and P. R. Schneider, "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits," *IEEE Transactions on Computers*, Vol. C-16, pp 567–579, October 1967.

[22] Michael H. Schulz, Erwin Trischler and Thomas M. Sarfert, "Socrates—A Highly Efficient Automatic Test Pattern Generation System," in *Proceedings of the 1987 International Test Conference,* pp 1016–1026, 1987.

[23] R. Tarjan, "Finding Dominators in Directed Graphs," *SIAM Journal of Computing*, Vol. 3, pp 62–89, 1974.

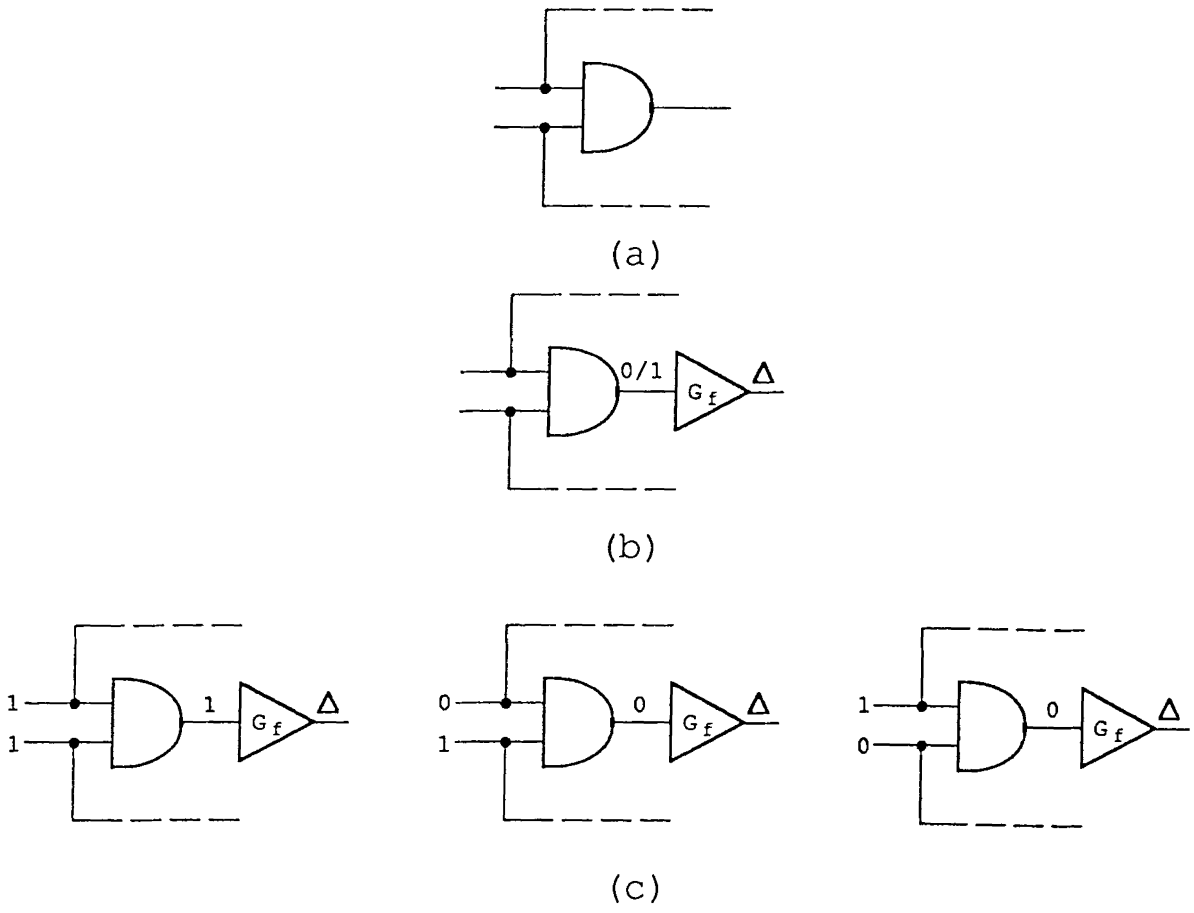

Figure 1: Common requirements for testing several checkpoint faults

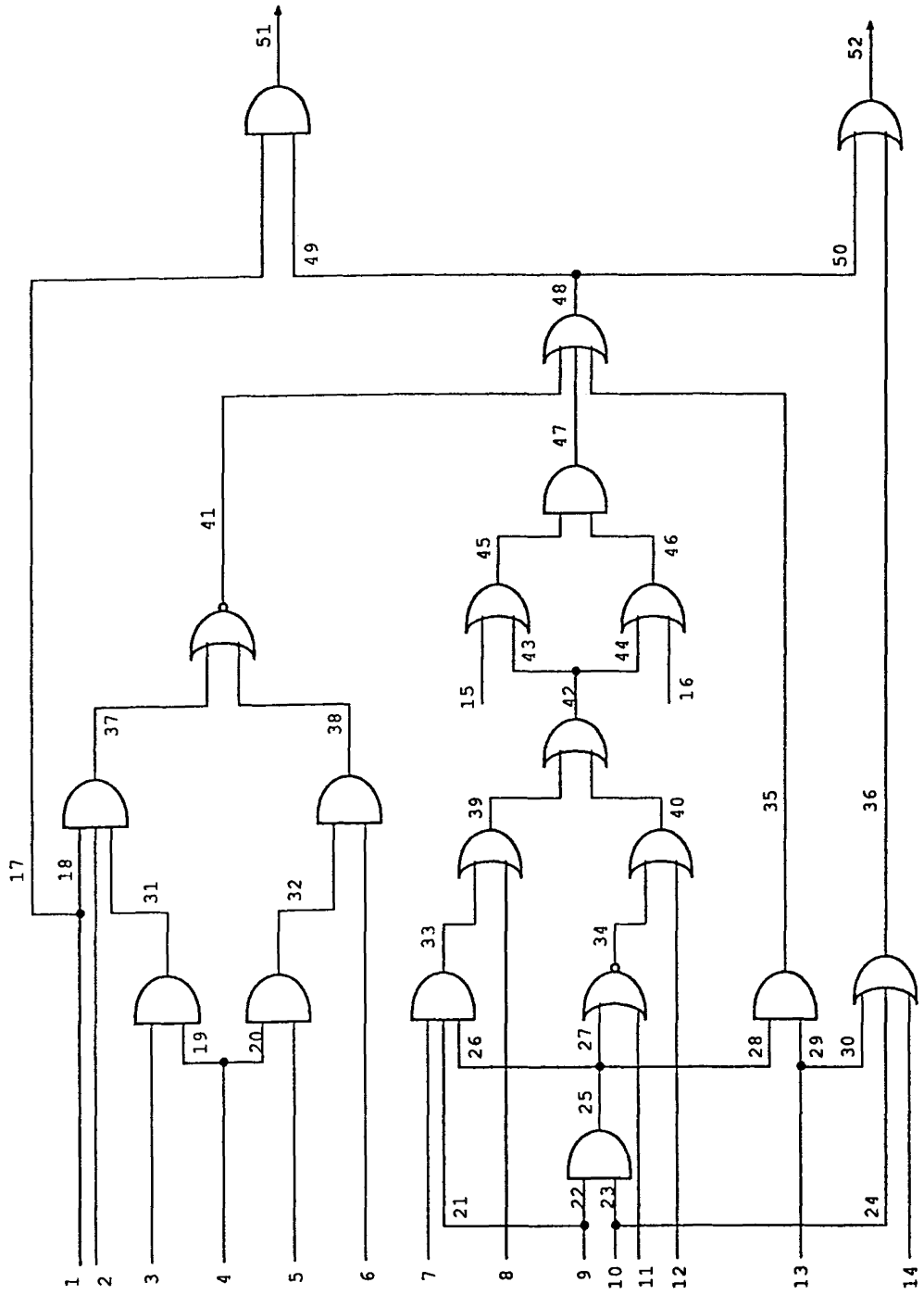

Figure 2: An example circuit

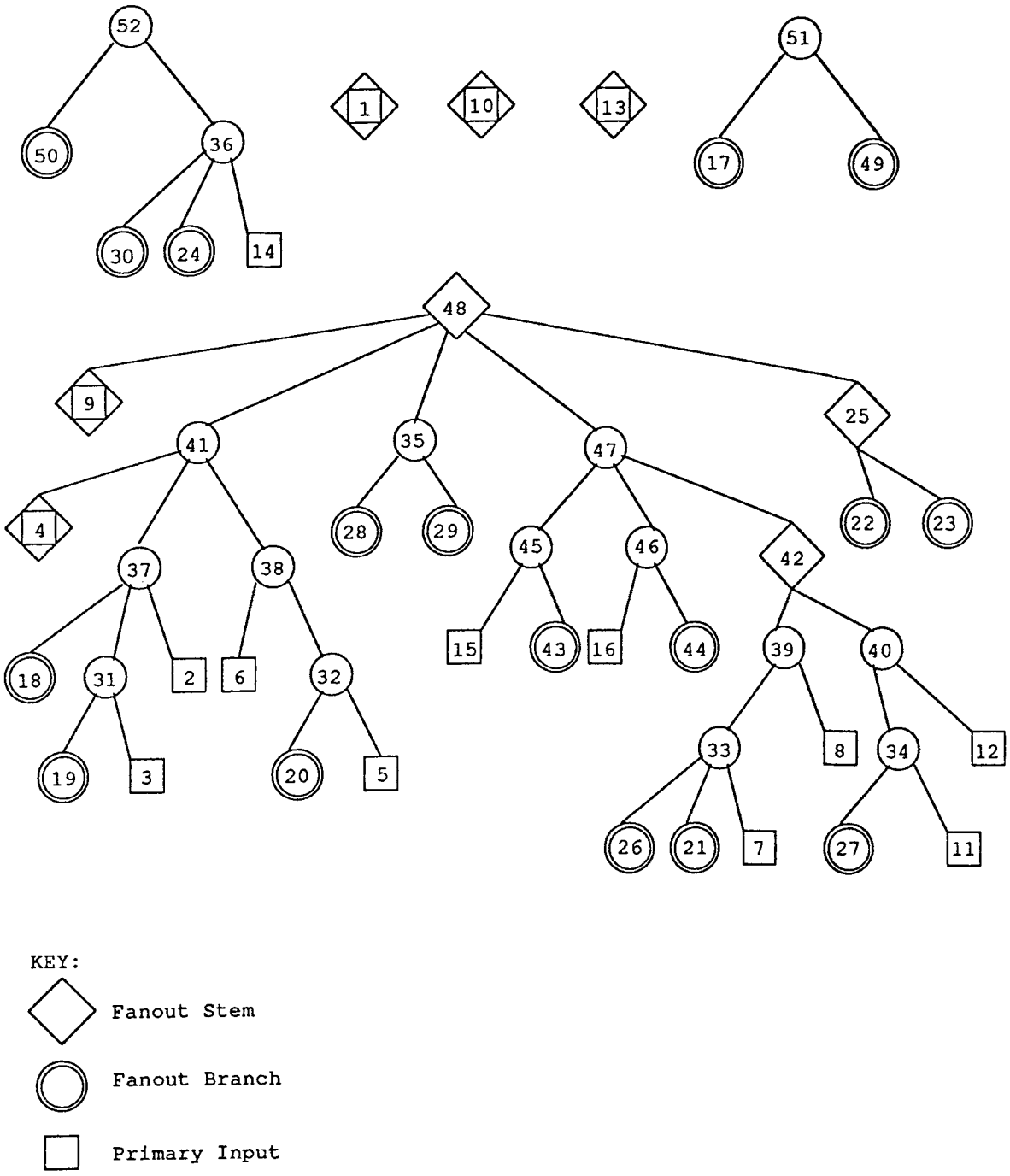

Figure 3: Dominator forest for example circuit

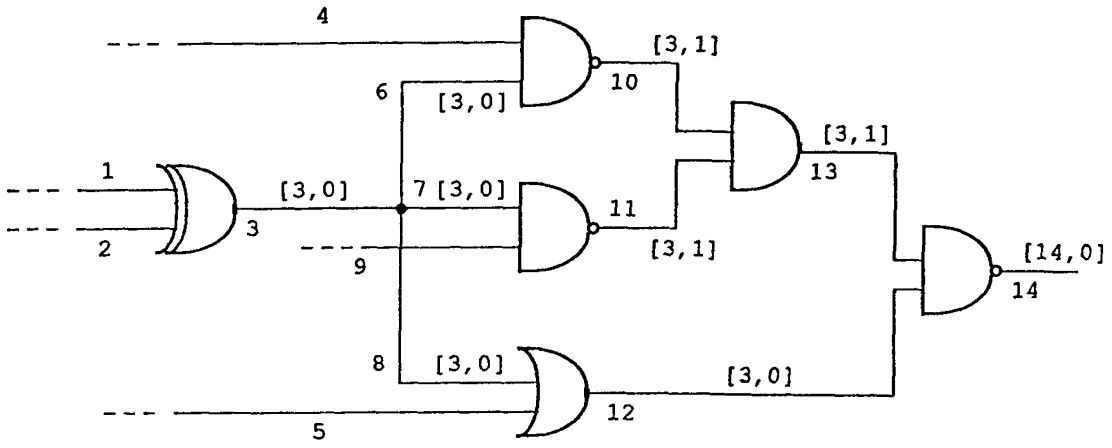

Figure 4: Use of token vectors

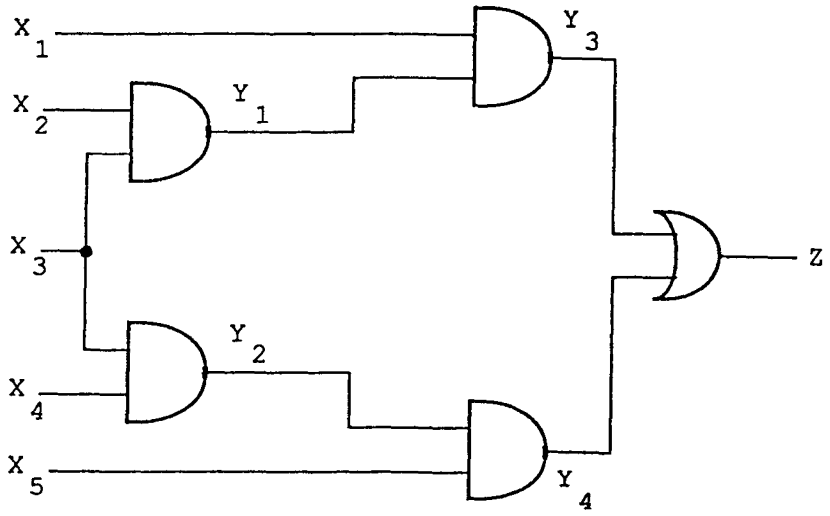

Figure 5: Use of the contrapositive

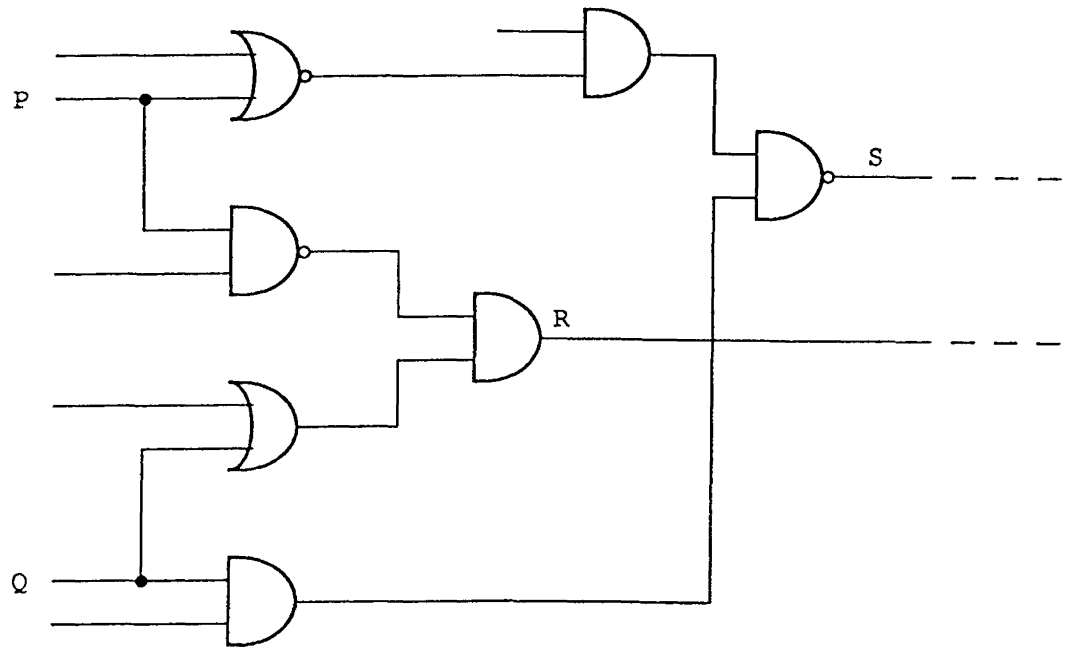

Figure 6: Example for contrapositive of a backward implication

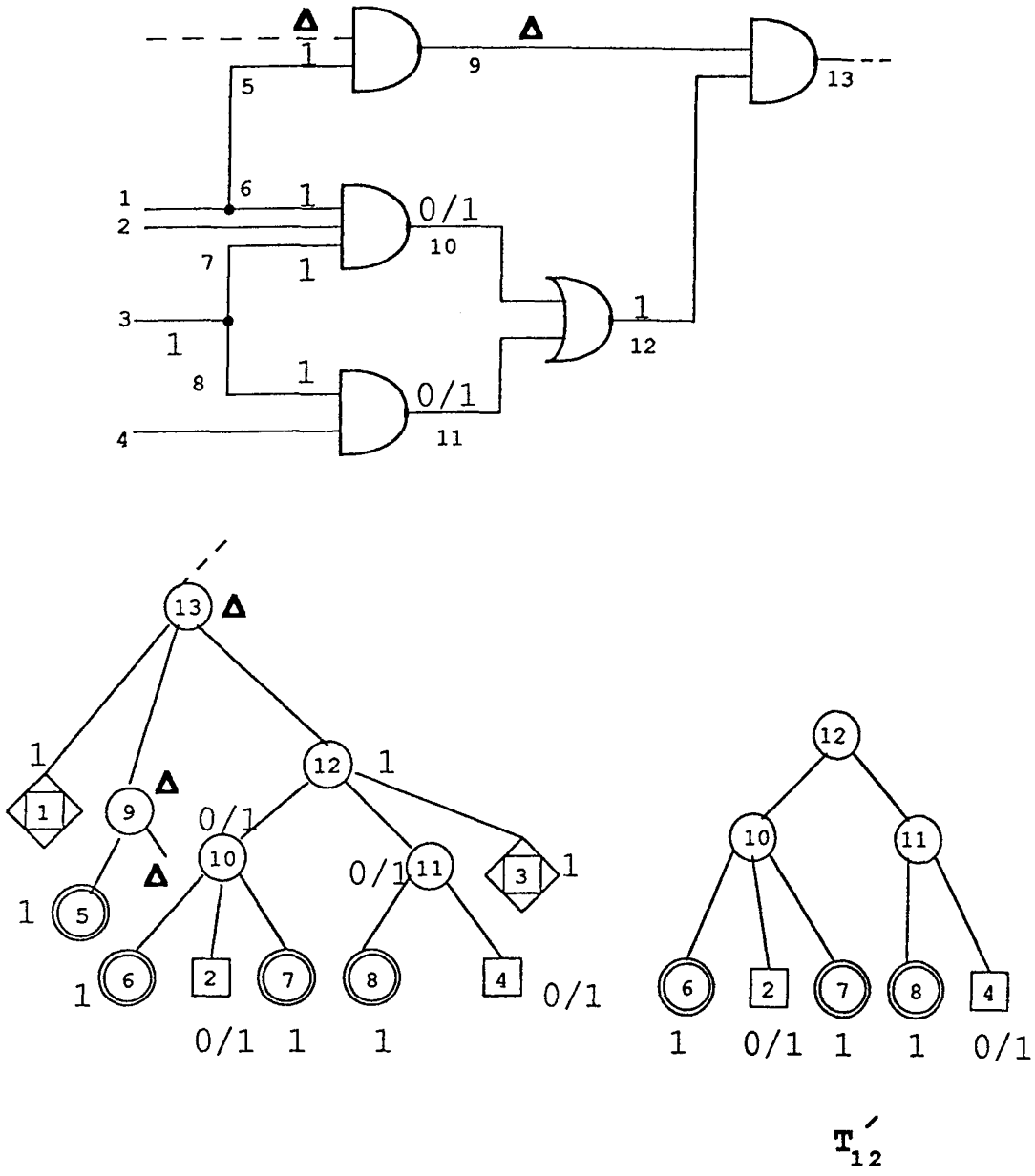

Figure 7: Identification of an SVN

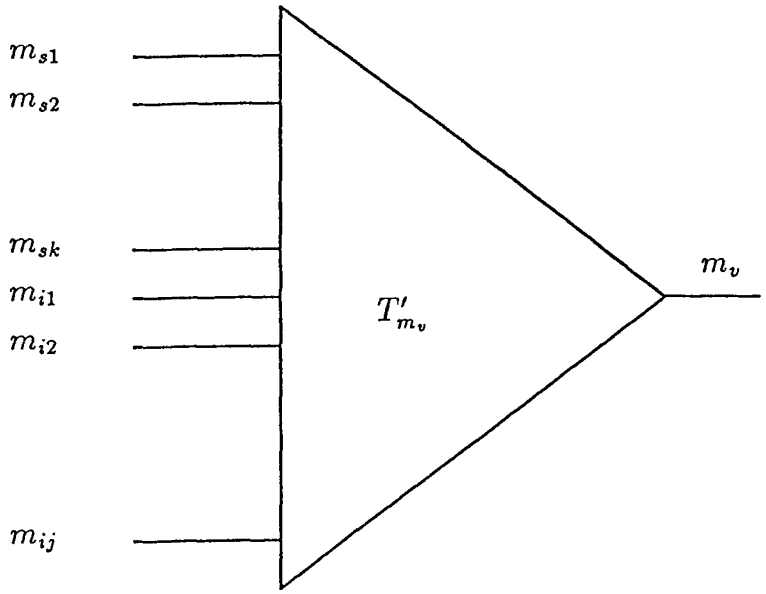

Figure 8: General structure of the subcircuit corresponding to the tree T'_{m_v}

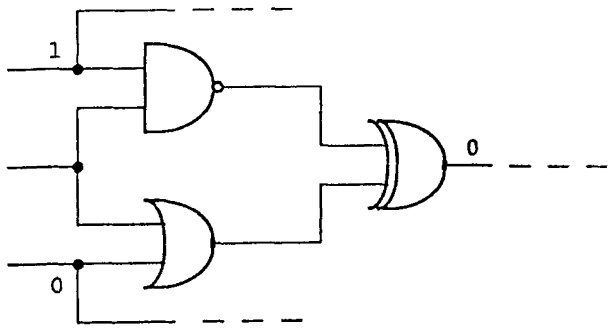

Figure 9: Unsatisfiable value at an IVN

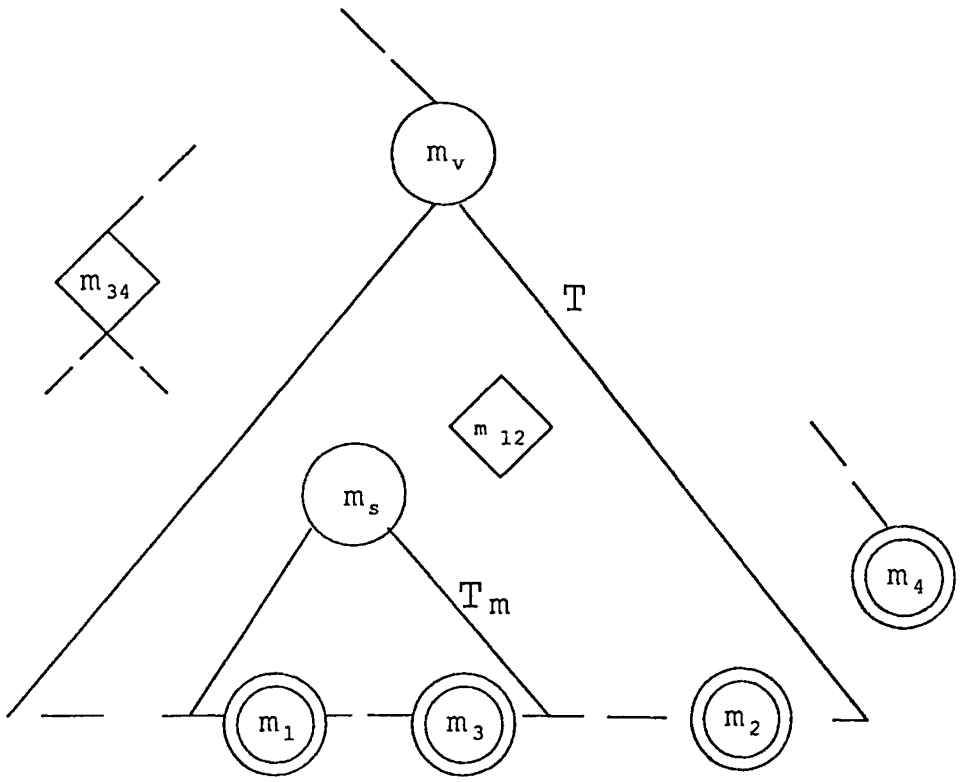

Figure 10: IVN identification example where $\mathcal{T}_{m_s}$ cannot be deleted.

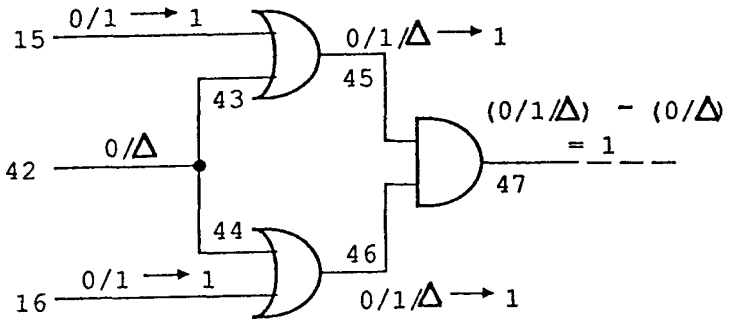

Figure 11: Backward implication of desensitizing values

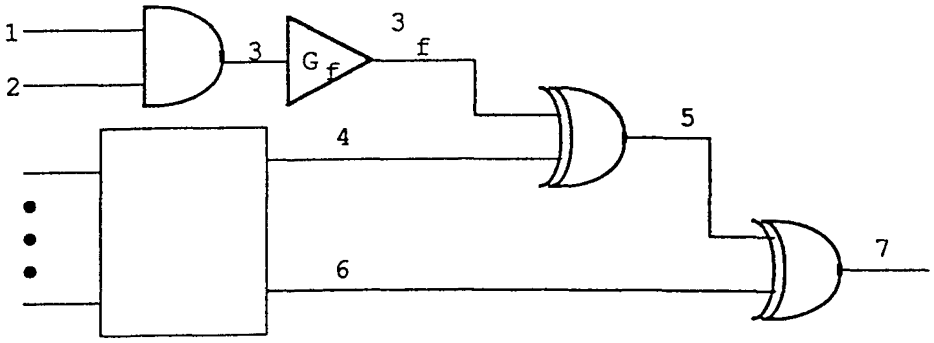

Figure 12: Example ECAT circuit

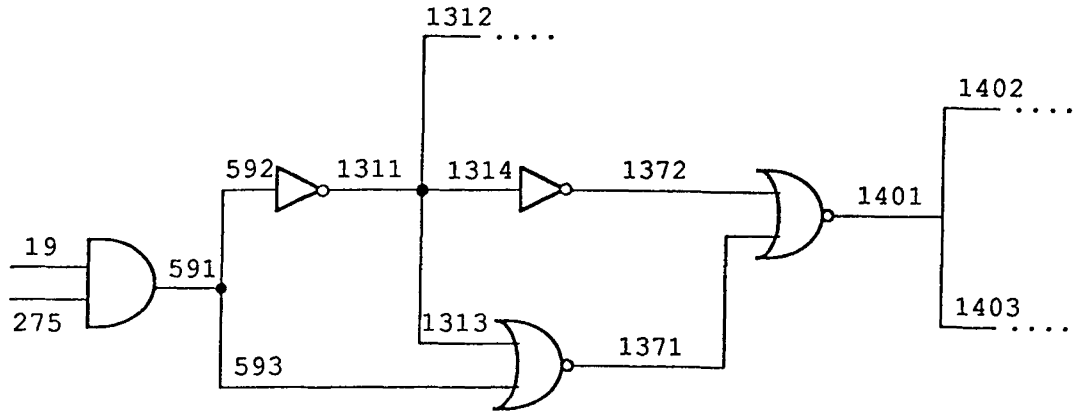

Figure 13: Use of the off dominator sensitizing inputs - I

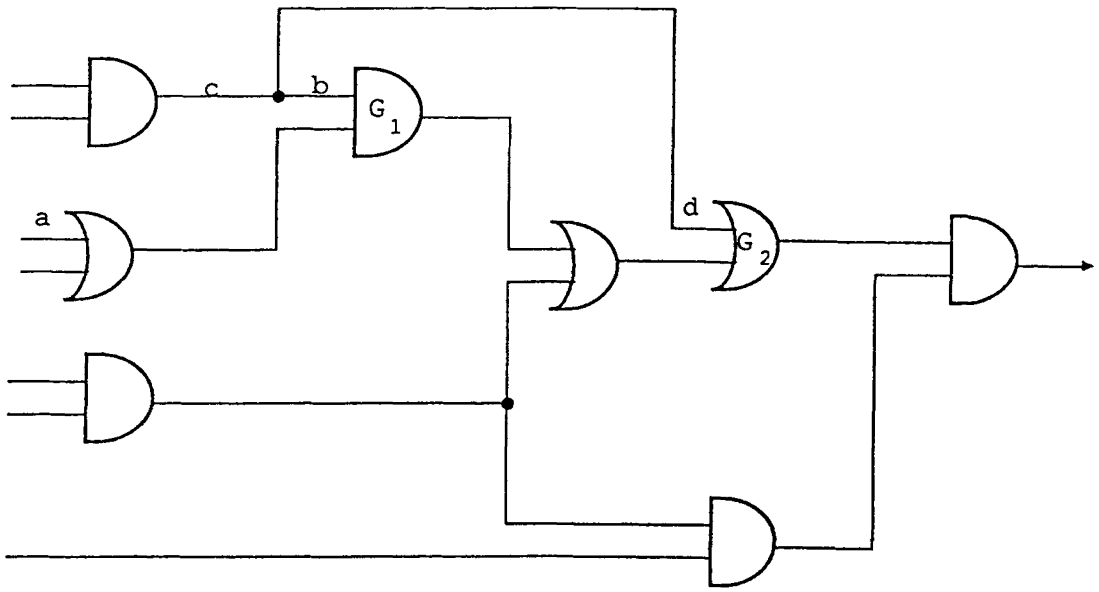

Figure 14: Use of the off dominator sensitizing inputs - II

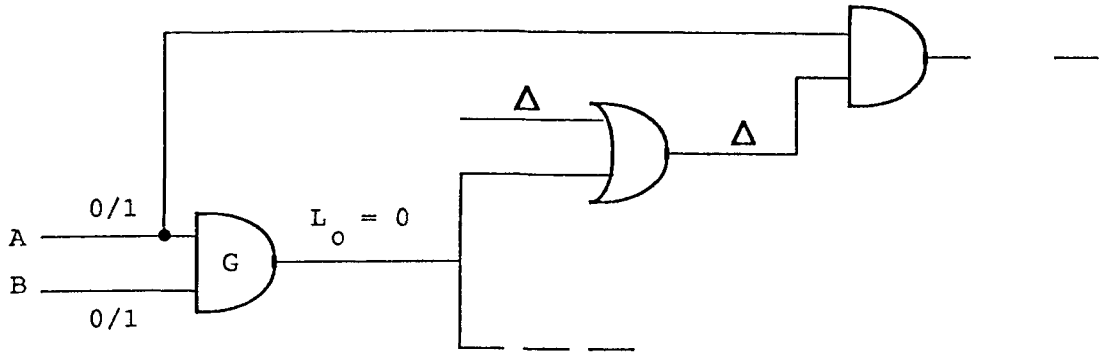

Figure 15: Example where $L_O \subset L_N$ during forward implication

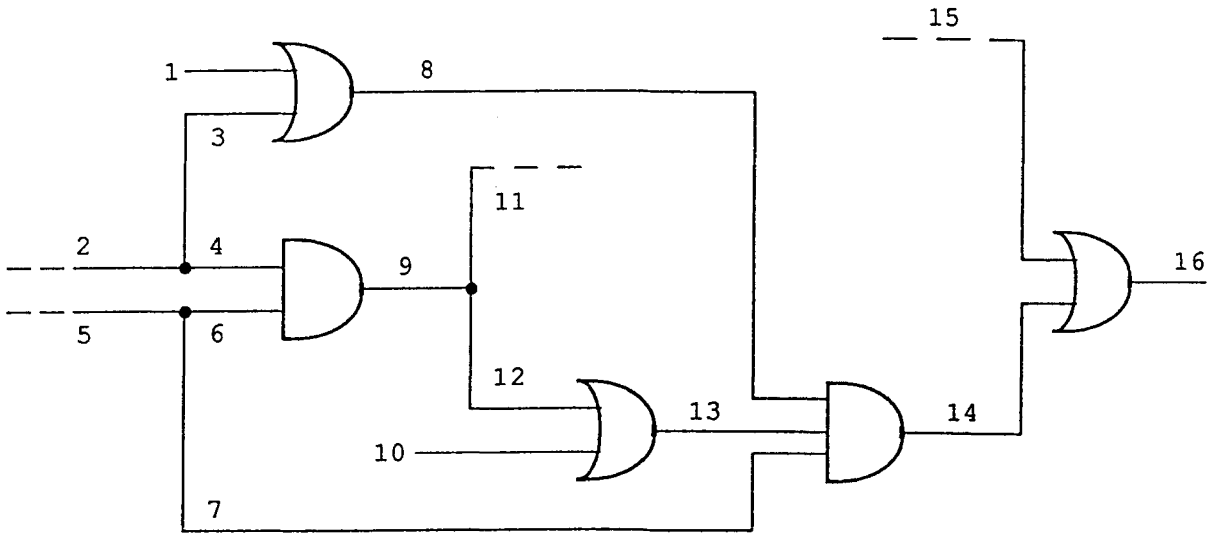

Figure 16: Example where $L_O \not\subseteq L_N$ and $L_N \not\subseteq L_O$ during forward implication

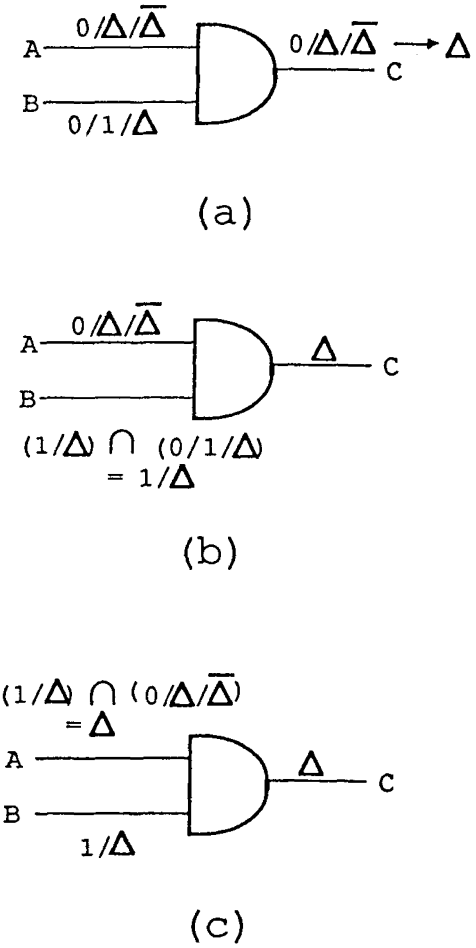

Figure 17: Example for backward implication

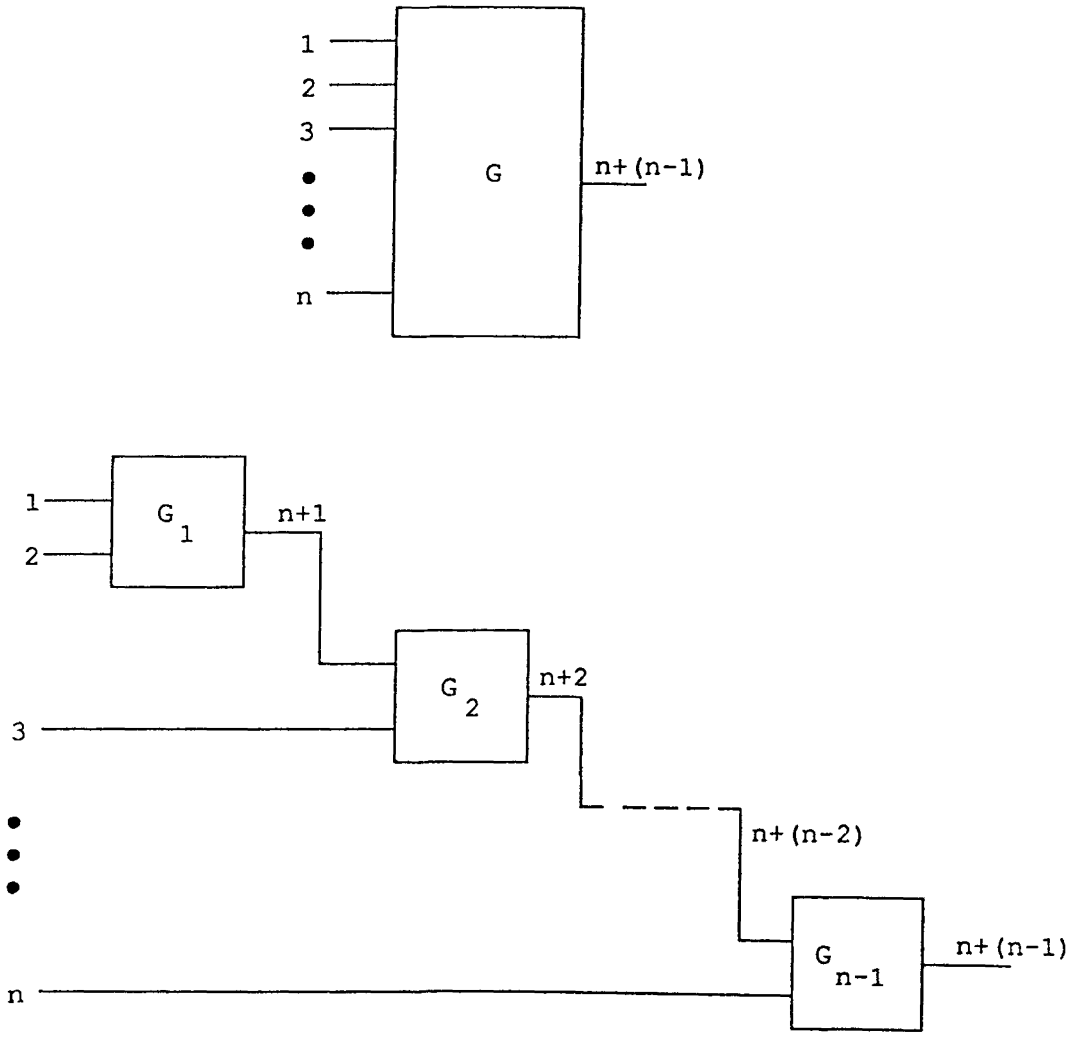

Figure 18: Gate decomposition

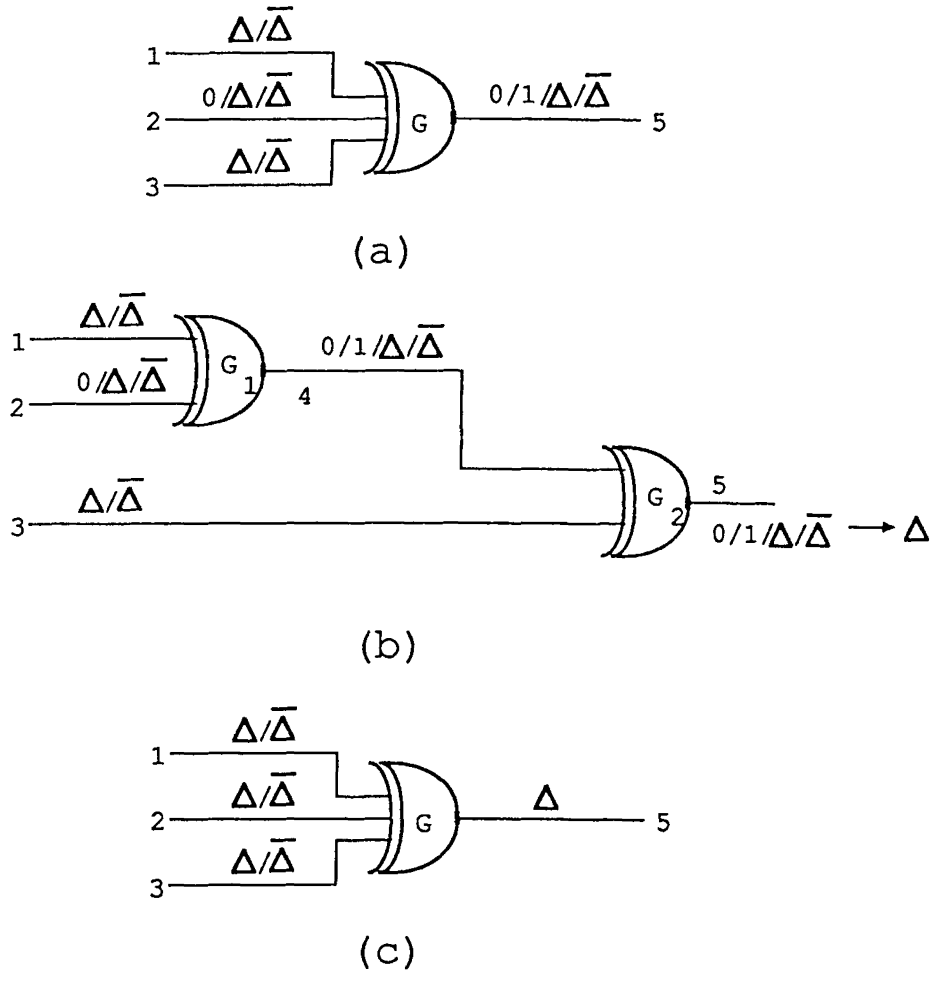

Figure 19: Example of backward implication for an XOR gate