An Exact Solution to the Minimum Size Test Pattern Problem

PAULO F. FLORES, HORÁCIO C. NETO and JOÃO P. MARQUES-SILVA
Technical University of Lisbon, IST/INESC

This article addresses the problem of test pattern generation for single stuck-at faults in combinational circuits, under the additional constraint that the number of specified primary input assignments is minimized. This problem has different applications in testing, including the identification of "don't care" conditions to be used in the synthesis of Built-In Self-Test (BIST) logic. The proposed solution is based on an integer linear programming (ILP) formulation which builds on an existing Propositional Satisfiability (SAT) model for test pattern generation. The resulting ILP formulation is linear on the size of the original SAT model for test generation, which is linear on the size of the circuit. Nevertheless, the resulting ILP instances represent complex optimization problems, that require dedicated ILP algorithms. Preliminary results on benchmark circuits validate the practical applicability of the test pattern minimization model and associated ILP algorithm.

Categories and Subject Descriptors: B.8.1 [**Performance and Reliability**]: *reliability, testing, and fault-tolerance*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Automatic test pattern generation (ATPG), built-in self-test (BIST), integer linear programming (ILP), propositional satisfiability (SAT), verification and test

1. INTRODUCTION

Automatic test pattern generation (ATPG) for stuck-at faults in combinational circuits is now a mature field, with an impressive number of highly effective models and algorithms [Goel 1981; Fujiwara and Shimono 1983; Kirkland and Mercer 1987; Schulz and Auth 1989; Giraldi and Bushnell 1991; Kunz and Pradham 1992; Larrabee 1992; Chakradhar et al. 1993; Lee and Ha 1993; Teramoto 1993; Cox and Rajski 1994; Silva and Sakallah 1994; Stephan et al. 1996]. Furthermore, besides being effective at detecting the target faults, recent ATPG tools have targeted the heuristic minimization (i.e., compaction) of the total number of test patterns required for detecting all faults in a circuit [Schulz and Auth 1989; Niermann and Patel 1991; Pomeranz et al. 1993; Hellebrand et al. 1995; Chakrabarty et al. 1997]. In general, the degree of test pattern compaction is expected to be related to the number of unspecified input assignments in each test pattern. In addition, for applications in which testing time

Authors' address: Technical University of Lisbon, IST/INESC, Rua Alves Redol N9, 1000-029 Lisboa, Portugal, e-mail: {pff, hcn, jpms}@inesc.pt.

and fault coverage requirements can only be obtained with dedicated Finite-State Machine (FSM) controllers, the computation of test patterns with a large number of unspecified input assignments may allow for significantly smaller synthesized FSMs. Indeed, if the test set is used as input to a logic synthesis tool with the purpose of synthesizing BIST logic, then by maximizing the number of unspecified input assignments, that is, by maximizing the "don't care" set of each test pattern, the logic synthesis tool is in general able to yield smaller synthesized logic. Thus, the maximization of the "don't care" set of each test pattern, or conversely, the computation of test patterns of minimum-size, can have significant practical advantages.

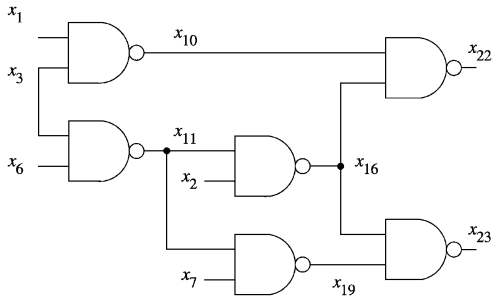
Nevertheless, there exists no model or algorithm in the literature for computing test patterns for which the number of unspecified primary input assignments is maximized. Accordingly, the main objective of this paper is to propose a first attempt at solving this problem. We start by formalizing the notion of test pattern minimization. We then develop a new model for test pattern generation, based on propositional satisfiability (SAT), in the presence of unspecified input assignments. Next, we derive an integer linear programming (ILP) model for maximizing the number of unspecified primary input assignments. Afterwards, we show that the model is indeed correct and analyze some of its limitations. Finally, we provide preliminary results that justify using the proposed model in medium-size combinational circuits and describe an ATPG methodology, which can incorporate the proposed model and supporting algorithm, and which can also be applied to large-size combinational circuits. Besides its practical applicability, to our best knowledge this is the first formal non-heuristic model towards computing minimum size test patterns.

The paper is organized as follows: We start in Section 2 with several definitions regarding combinational circuits, Conjunctive Normal Form (CNF) representations of circuits and CNF representations of fault detection problems, which are used throughout the paper. Afterwards, in Section 3, the CNF models described in Section 2 are generalized for correctly handling unspecified variable assignments. The next step is to introduce the ILP optimization model for minimizing test patterns and prove its correctness. Section 5 includes preliminary experimental results on several practical applications of the model. We conclude in Section 6 with a brief overview of future research work in the area of test pattern minimization.

2. DEFINITIONS

2.1 Combinational Circuits

We start by introducing unified representations for circuits and fault detection problems. These representations are used throughout the article. A combinational circuit C is represented as a directed acyclic graph $C=(V_C, E_C)$, where the elements of V_C, that is, the circuit nodes, are either primary inputs or gate outputs, with $|V|=N$. The set of edges $E_C \subseteq V_C \times V_C$ identifies gate input–output connections. We shall assume gates with bounded fanin, and so $|E_C|=O(|N|)$. For every circuit node x in V, the following definitions apply:



Node x	x_{11}
$O(x)$	$\{x_{16}, x_{19}\}$
$O^*(x)$	$\{x_{16}, x_{19}, x_{22}, x_{23}\}$
$I(x)$	$\{x_3, x_6\}$
$I^*(x)$	$\{x_3, x_6\}$
$K_O(x)$	$\{x_2, x_7, x_{10}, x_{16}, x_{19}, x_{22}, x_{23}\}$
$K_I(x)$	$\{x_{10}, x_2, x_7, x_1, x_3, x_6\}$

Fig. 1. Example circuit, c17, and topological data for node x_{11}.

—$O(x)$ denotes the *fanout* nodes of node x, that is, nodes y in V_C such that $(x, y) \in E_C$.

—$O^*(x)$ denotes the *transitive fanout* of node x, that is, the set of all nodes y such that there is a path connecting x to y.

—$I(x)$ denotes the *fanin* nodes of node x, that is, nodes y in V_C such that $(y, x) \in E_C$.

—$I^*(x)$ denotes the *transitive fanin* of node x, that is, the set of all nodes y such that there is a path connecting y to x.

—$K_O(x)$ denotes *immediate fanout cone of influence* of x, being defined as follows:

$$K_O(x) = \{y \mid y \in O^*(x) \vee y \in I(w) \wedge w \in O^*(x)\}. \tag{1}$$

—$K_I(x)$ denotes *immediate fanin cone of influence* of x, being defined as follows:

$$K_I(x) = \left[\bigcup_{y \in O^*(x)} I^*(y)\right] - (O^*(x) \cup \{x\}). \tag{2}$$

The set of primary inputs can also be referred to as PI, and the set of primary outputs as PO. Simple gates are assumed: AND, NAND, OR, NOR, NOT and BUFF. Finally, the number of stuck-at faults in the circuit is M, with $M = O(N)$, since $|E_C| = O(|N|)$, and are numbered $1, \ldots, M$. The example in Figure 1 illustrates the previous definitions.

2.2 Conjunctive Normal Form Formulas

A conjunctive normal form (CNF) formula φ on n binary variables $x_1, \ldots, x_n$ is the conjunction (AND) of m *clauses* $\omega_1, \ldots, \omega_m$ each of which is the disjunction (OR) of one or more *literals*, where a literal is the occurrence of a variable x_i or its complement $\neg x_i$. A formula φ denotes a unique n-variable Boolean function $f(x_1, \ldots, x_n)$ and each of its clauses corresponds to an implicate of f. An *assignment* for a formula φ is a set of variables and their corresponding Boolean values, represented as variable/value pairs; for example, $A = \{(x_1, 0), (x_7, 1), (x_{13}, 0)\}$. Alternatively, assignments can be denoted as $A = \{x_1 = 0, x_7 = 1, x_{13} = 0\}$.

The CNF formula of a circuit is the conjunction of the CNF formulas for each gate output, where the CNF formula of each gate denotes the valid input–output assignments to the gate. For an AND gate, $x = \text{AND}(w_1, \ldots, w_j)$ the resulting CNF formula is [Larrabee 1992; Stephan et al. 1996; Silva and Sakallah 1997],

$$\varphi_x = \left[\prod_{i=1}^{j}(w_i + \neg x)\right] \cdot \left(\sum_{i=1}^{j} \neg w_i + x\right). \tag{3}$$

A complete list of the CNF formulas for simple gates with an arbitrary number of inputs can be found in Silva and Sakallah [1997]. If we view a CNF formula as a set of clauses, the CNF formula for the circuit is defined by the set union of the CNF formulas for each gate with output x, φ_x:

$$\varphi = \bigcup_{x \in V_C} \varphi_x. \tag{4}$$

2.3 Test Pattern Generation

For Automatic Test Pattern Generation (ATPG), the single stuck-at line (SSF) fault model is assumed, and the following definitions apply.[1]

Definition 1. We say that a stuck-at fault is *detectable* if and only if there exists an assignment of logic values to the circuit primary inputs such that the effect of the discrepancy caused by the fault (i.e., the error signal) can be observed on at least one of the circuit primary outputs (i.e., the value in the good and faulty circuit differ).

When referring to primary input assignments, or test patterns, we may in general assume that some primary inputs may be unspecified.

Definition 2. We define a test pattern T as an assignment to the primary inputs, such that some assignments may be unspecified, that is, $T = \{(x, v), x \in PI \wedge v \in \{0, 1, X\}\}$.

Definition 3. A test pattern T is *completely specified* whenever $T = \{(x, v), x \in PI \wedge v \in \{0, 1\}\}$. Otherwise, T is said to be *incompletely specified*.

In the remainder of this section, we shall assume that test patterns are completely specified. The generation of incompletely specified test patterns is addressed in Section 3.

CNF representations of circuits and fault detection problems have been extensively used and studied in ATPG [Larrabee 1992; Chakradhar et al. 1993; Stephan et al. 1996; Silva and Sakallah 1997]. In this section, we describe a simple CNF representation of combinational circuits and fault detection problems, which will be used throughout the remainder of the article.

In the context of test pattern generation, and for capturing the fault detection problem, each node x is characterized by three propositional variables:

[1]See Abramovici et al. [1990] for ATPG definitions used throughout the paper.

—x^{G} denotes the logic value assumed by the node in the *good* circuit.

—x^{F} denotes the logic value assumed by the node in the *faulty* circuit.

—x^{S} denotes whether x^{G} and x^{F} assume different logic value [Larrabee 1992]. We shall refer to this variable as the *sensitization status* of node x.

Given the definition of variable x^{S}, the following relationship must hold:

$$[(x^{\mathrm{G}} \neq x^{\mathrm{F}}) \leftrightarrow x^{\mathrm{S}}] \Leftrightarrow (x^{\mathrm{G}} + \neg x^{\mathrm{F}} + x^{\mathrm{S}}) \cdot (\neg x^{\mathrm{G}} + x^{\mathrm{F}} + x^{\mathrm{S}}) \cdot (x^{\mathrm{G}} + x^{\mathrm{F}} + \neg x^{\mathrm{S}}) \cdot (\neg x^{\mathrm{G}} + \neg x^{\mathrm{F}} + \neg x^{\mathrm{S}}), \quad (5)$$

which basically states that the logic values of x^{G} and x^{F} differ if and only if x^{S} assumes logic value 1.

Let φ_x denote the CNF formula associated with gate output x. The notation φ_x^{G} denotes the CNF formula for x in the good circuit, that is, using x^{G} variables, whereas φ_x^{F} denotes the CNF formula for x in the faulty circuit, that is, using x^{F} variables. For a *stem* fault z s-a-v, the CNF representation of the associated fault detection problem contains the following components:

—CNF formula denoting the good circuit.

—CNF formula denoting the faulty circuit. This formula only needs to contain the CNF formulas for the nodes that are relevant for detecting the given fault, that is, nodes in the transitive fanout of node z.

—CNF formulas for defining the sensitization status of every node in the transitive fanout of the fault site, that is, node z. Hence, for each of these nodes, φ_x^{S}, is given by (5), which requires $x^{\mathrm{S}} = 1$ if and only if $x^{\mathrm{G}} \neq x^{\mathrm{F}}$.

—Clauses that prevent each node x from being sensitized, by having $x^{\mathrm{S}} = 0$, whenever x is not in the transitive fanout of z but at least one fanout node of x is in the transitive fanout of z, that is, $x \in K_O(z) - O^*(z)$. (Observe that this condition on a node x also implies $x^{\mathrm{G}} = x^{\mathrm{F}}$. Moreover, this condition permits reducing the number of x^{F} and x^{S} variables that must be considered.)

—Clauses requiring $x^{\mathrm{G}} = x^{\mathrm{F}}$ on each node x such that x is not in the transitive fanout of z but at least one fanout node of x is in the transitive fanout of z, that is, x is in $K_O(z) - O^*(z)$. (Observe that this condition and the previous one permit restricting the number of x^{F} and x^{S} variables that must actually be used.)

—Clauses capturing conditions for *activating* the fault, that is, by requiring $z^{\mathrm{G}} \neq z^{\mathrm{F}}$ and by forcing a suitable logic value on z^{G}.

—Finally, we guarantee that the fault effect is observed at a primary output by requiring that for at least one primary output x, $x^{\mathrm{S}} = 1$.

The formula for detecting a fault z s-a-v is summarized in Table I and will henceforth be referred to as the *fault detection* formula, φ^{D}.

The CNF formula for fanout–branch faults can be similarly defined [Silva and Sakallah 1997]. In addition, the model described above can be improved with additional clauses which further constrain the problem definition [Larrabee 1992; Chakradhar et al. 1993; Silva and Sakallah 1994; Stephan et al. 1996].

Table I. Definition of the Fault Detection Problem for the Stem Fault z s-a-v

Sub-formula/Condition	Clause Set
Good Circuit	$\varphi^{\mathrm{G}} = \bigcup_{x \in V_C} \varphi_x^{\mathrm{G}}$
Faulty Circuit	$\varphi^{\mathrm{F}} = \bigcup_{x \in O^*(z)} \varphi_x^{\mathrm{F}}$
Node Sensitization	$\varphi^{\mathrm{S}} = \bigcup_{x \in O^*(z)} \varphi_x^{\mathrm{S}}$
Propagation Blocking Conditions	$\varphi^{\mathrm{B}} = (\neg x^{\mathrm{S}}) \qquad x \in K_O(z) - O^*(z)$
Fault Activation Conditions	$\varphi^{\mathrm{A}} = \begin{cases} (z^{\mathrm{S}}) \cdot (\neg z^{\mathrm{G}}) \cdot (z^{\mathrm{F}}) & \text{if } v = 1 \\ (z^{\mathrm{S}}) \cdot (z^{\mathrm{G}}) \cdot (\neg z^{\mathrm{F}}) & \text{if } v = 0 \end{cases}$
Fault Detection Requirement	$\varphi^{\mathrm{R}} = \left(\sum_{x \in PO \wedge x \in O^*(z)} x^{\mathrm{S}} \right)$
Detection of Fault z s-a-v	$\varphi^{\mathrm{D}} = \varphi^{\mathrm{G}} \cup \varphi^{\mathrm{F}} \cup \varphi^{\mathrm{S}} \cup \varphi^{\mathrm{B}} \cup \varphi^{\mathrm{A}} \cup \varphi^{\mathrm{R}}$

3. TEST GENERATION WITH UNSPECIFIED VARIABLE ASSIGNMENTS

The SAT-based test generation model described in the previous section requires all clauses to be satisfied; hence, most if not all variables must be assigned a logic value. However, we want to develop a test generation model that properly handles unspecified variable assignments, since our goal is to compute minimum size test patterns. As a result, in this section, we develop models for circuit satisfiability and test generation using CNF formulas that can be satisfied in the presence of unspecified variable assignments.

3.1 Modeling Unspecified Variable Assignments

Given a circuit and its associated CNF formula or a fault f and its associated fault detection formula, the existence of unspecified assignments implies that each of the original circuit variables can now be assigned a value in the set $\{0, 1, X\}$. In this situation an assignment $x = X$ indicates that x is *unspecified*, or that the value assumed by x is an unspecified assignment.[2] This signifies that an assignment A is now allowed to leave variables unspecified.

With the purpose of deciding CNF formula satisfiability, in the presence of unspecified variables, a new set of variables is created. This basically consists of duplicating the number of Boolean variables, which is a common solution

[2]Note that $x \in \{0, 1\}$ indicates that x is *specified*, or that the value assumed by x is a specified assignment.

Table II. Interpretation of the New Variables Modeling Unspecified Assignments

x	(x^1, x^0)
0	(0, 1)
1	(1, 0)
X	(0, 0)

for capturing unspecified assignments [Pizzuti 1996].[3] As a result, we propose to represent each Boolean variable x with two new variables x^0 and x^1 having the interpretation indicated in Table II. For this interpretation, $x = X$ indicates that x is unspecified. The simultaneous assignment of variables x^0 and x^1 to 1 is not allowed, requiring the inclusion of the following constraint in the resulting CNF formula,

$$\varphi_{inv,x} = (\neg x^1 + \neg x^0) \tag{6}$$

for each node $x \in V_C$, where V_C represents the set of nodes in the circuit.

In addition, for each basic gate type, we need to define the corresponding CNF formula. However, using the ideas above, each gate input and output must now be replaced by two variables. Let us consider for example an AND gate, which will now be denoted by the generalized form $(x^0, x^1) = U\,AND(w_1^0, w_1^1, \ldots, w_j^0, w_j^1)$, and which allows unspecified assignments to the gate inputs and output. Since the simultaneous assignment of any pair of variables (x^0, x^1) to 1 is prevented by (6), then we just need to relate the remaining assignments. The output variable x^1 can only assume value 1 whenever all input variables w_j^1 also assume value 1. Hence, we can say that $x^1 = AND(w_1^1, \ldots, w_j^1)$. In addition, the output variable x^0 assumes value 1 provided at least one input variable w_j^0 assumes value 1. Hence, we can say that $x^0 = OR(w_1^0, \ldots, w_j^0)$. As a result, we obtain from Larrabee [1992] and Silva and Sakallah [1997],

$$\begin{aligned} \varphi_{u,x^0} &= \left[\prod_{i=1}^{j} \left(\neg w_i^0 + x^0\right)\right] \cdot \left(\sum_{i=1}^{j} w_i^0 + \neg x^0\right) \\ \varphi_{u,x^1} &= \left[\prod_{i=1}^{j} \left(w_i^1 + \neg x^1\right)\right] \cdot \left(\sum_{i=1}^{j} \neg w_i^1 + x^1\right). \end{aligned} \tag{7}$$

Furthermore, the CNF formula for an AND gate with output x now becomes,

$$\varphi_{u,x} = \varphi_{u,x^1} \bigcup \varphi_{u,x^0} \bigcup \varphi_{inv,x}, \tag{8}$$

which properly models unspecified assignments to the inputs and output of an AND gate. Similar relations can be derived for the other simple gates. Consequently, the CNF formulas for simple gates given in Silva and Sakallah [1997]

[3]Observe that since only 3^M assignments need to be considered for M variables, the actually required number of Boolean variables is $\lceil \log(3^M) \rceil$. Nevertheless, considering instead $2M$ variables greatly simplifies the proposed model.

Table III. Generalized CNF Formulas for Simple Gates

Gate type	Gate function	φ_{u,x^i}
AND	$x^0 = OR\left(w_1^0, \ldots, w_j^0\right)$	$\left[\prod_{i=1}^{j}\left(\neg w_i^0 + x^0\right)\right] \cdot \left(\sum_{i=1}^{j} w_i^0 + \neg x^0\right)$
	$x^1 = AND\left(w_1^1, \ldots, w_j^1\right)$	$\left[\prod_{i=1}^{j}\left(w_i^1 + \neg x^1\right)\right] \cdot \left(\sum_{i=1}^{j} \neg w_i^1 + x^1\right)$
NAND	$x^0 = AND\left(w_1^1, \ldots, w_j^1\right)$	$\left[\prod_{i=1}^{j}\left(w_i^1 + \neg x^0\right)\right] \cdot \left(\sum_{i=1}^{j} \neg w_i^1 + x^0\right)$
	$x^1 = OR\left(w_1^0, \ldots, w_j^0\right)$	$\left[\prod_{i=1}^{j}\left(\neg w_i^0 + x^1\right)\right] \cdot \left(\sum_{i=1}^{j} w_i^0 + \neg x^1\right)$
OR	$x^0 = AND\left(w_1^0, \ldots, w_j^0\right)$	$\left[\prod_{i=1}^{j}\left(w_i^0 + \neg x^0\right)\right] \cdot \left(\sum_{i=1}^{j} \neg w_i^0 + x^0\right)$
	$x^1 = OR\left(w_1^1, \ldots, w_j^1\right)$	$\left[\prod_{i=1}^{j}\left(\neg w_i^1 + x^1\right)\right] \cdot \left(\sum_{i=1}^{j} w_i^1 + \neg x^1\right)$
NOR	$x^0 = OR\left(w_1^1, \ldots, w_j^1\right)$	$\left[\prod_{i=1}^{j}\left(\neg w_i^1 + x^0\right)\right] \cdot \left(\sum_{i=1}^{j} w_i^1 + \neg x^0\right)$
	$x^1 = AND\left(w_1^0, \ldots, w_j^0\right)$	$\left[\prod_{i=1}^{j}\left(w_i^0 + \neg x^1\right)\right] \cdot \left(\sum_{i=1}^{j} \neg w_i^0 + x^1\right)$
NOT	$x^0 = BUFF\left(w_1^1\right)$	$\left(w_1^1 + \neg x^0\right) \cdot \left(\neg w_1^1 + x^0\right)$
	$x^1 = BUFF\left(w_1^0\right)$	$\left(w_1^0 + \neg x^1\right) \cdot \left(\neg w_1^0 + x^1\right)$
BUFF	$x^0 = BUFF\left(w_1^0\right)$	$\left(w_1^0 + \neg x^0\right) \cdot \left(\neg w_1^0 + x^0\right)$
	$x^1 = BUFF\left(w_1^1\right)$	$\left(w_1^1 + \neg x^1\right) \cdot \left(\neg w_1^1 + x^1\right)$

can be generalized by following the same approach used for deriving (7). These generalized CNF formulas are given in Table III. As a result, and as done in Section 2.2, we can now create the CNF formula for the circuit in which unspecified variable assignments are allowed. In Figure 2, we illustrate the outcome of applying an incompletely specified assignment to the primary inputs of c17. As

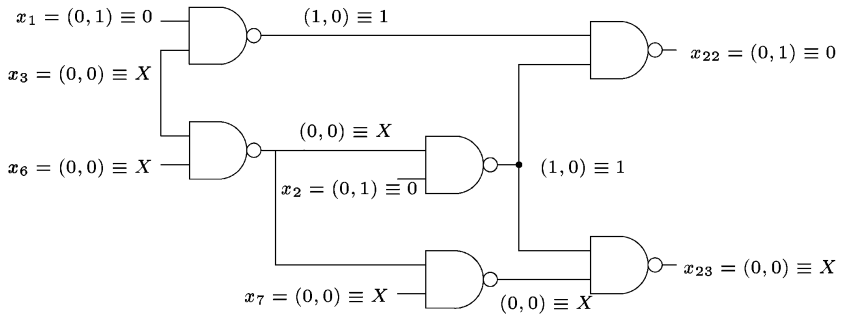


Fig. 2. Example of unspecified assignments for c17.

can be seen, the assignments $x_1 = 0$ and $x_2 = 0$ represent a sufficient condition for the assignment $x_{22} = 0$ to be observed.

3.2 Test Pattern Generation with Unspecified Input Assignments

We can now generalize the test pattern generation model of Section 2.3 so that unspecified variable assignments are allowed. Each circuit node x is still characterized by three variables:

—x^{G} denoting the value in the good circuit. This variable can be unspecified, and so we use two new variables to characterize its value, $x^{\mathrm{G},0}$ and $x^{\mathrm{G},1}$, with the semantic definition given earlier.

—x^{F} denoting the value in the faulty circuit. This variable can also be unspecified, and so we use two new variables to characterize its value, $x^{\mathrm{F},0}$ and $x^{\mathrm{F},1}$, with the semantic definition given earlier.

—x^{S} denoting the sensitization status of each node. As we justify in this article, the sensitization status of each node needs not be unspecified, and so its value is always either 0 or 1.

Modeling unspecified assignments in test generation requires a detailed characterization of the propagation conditions of the fault effect. Hence, the sensitization status x^{S} of a node can only assume value 1 when both values of node in the good and faulty circuits are *specified* and assume different logic values. Moreover this requirement also causes the value of a node in the faulty circuit to be specified *only* when the value of that node in the good circuit is also specified. These constraints indicate that propagation of the fault effect to a node can only be guaranteed when the values in the good and faulty circuit are specified for that node.

Consequently, the relationship between the value of x^{S} and the possible values of x^{G} and x^{F} is shown in Table IV. Entries with a "–" denote invalid value assignments, for which the CNF formula for x^{S} must assume value 0. Similarly to the model for completely specified assignments, x^{S} assumes value 1 if and only if x^{G} and x^{F} assume opposing logic values, provided that both x^{G} and x^{F} are specified. The simplification of the truth Table IV yields the following CNF

Table IV. Truth Table for the Sensitization Status

x^G	x^F	x^S
X	X	0
0	0	0
0	1	1
1	0	1
1	1	0
1	X	0
0	X	0
X	0	–
X	1	–

formula for the sensitization status of node x, x^{S}:

$$\begin{aligned}\varphi^{\mathrm{S}}_{u,x} \;=\; & (x^{\mathrm{G},1} + x^{\mathrm{G},0} + \neg x^{\mathrm{S}}) \cdot (x^{\mathrm{F},1} + x^{\mathrm{F},0} + \neg x^{\mathrm{S}}) \cdot (x^{\mathrm{F},1} + x^{\mathrm{G},1} + \neg x^{\mathrm{S}}) \\ & \cdot (\neg x^{\mathrm{G},1} + \neg x^{\mathrm{F},1} + \neg x^{\mathrm{S}}) \cdot (x^{\mathrm{G},1} + \neg x^{\mathrm{F},1} + x^{\mathrm{S}}) \cdot (x^{\mathrm{G},0} + \neg x^{\mathrm{F},0} + x^{\mathrm{S}}). \quad (9)\end{aligned}$$

The next step is to describe the modifications to the CNF formula used for computing the faulty values, which for completely specified assignments are equivalent to the CNF formula for the good value. For incompletely specified assignments, the same holds true but, as justified above, we introduce the additional constraint that an unspecified good value implies and unspecified faulty value

$$(x^G = X) \Rightarrow (x^F = X). \quad (10)$$

Let us assume that the CNF formula for the faulty value of a node x with completely specified assignments is given by

$$\varphi^F_x = \prod_{i=1}^{j} \omega_i. \quad (11)$$

As a result of (10), the CNF formula for the faulty circuit, in the presence of incompletely specified assignments, is defined by

$$\begin{aligned}\varphi^{\mathrm{F}}_{u,x} \;=\; & (\neg x^{\mathrm{F},0} + x^{\mathrm{G},0} + x^{\mathrm{G},1}) \cdot (\neg x^{\mathrm{F},1} + x^{\mathrm{G},0} + x^{\mathrm{G},1}) \\ & \cdot \prod_{i=1}^{j} (\omega_i + \neg x^{\mathrm{G},0} \cdot \neg x^{\mathrm{G},1}) \quad (12) \\ =\; & (\neg x^{\mathrm{F},0} + x^{\mathrm{G},0} + x^{\mathrm{G},1}) \cdot (\neg x^{\mathrm{F},1} + x^{\mathrm{G},0} + x^{\mathrm{G},1}) \\ & \cdot \prod_{i=1}^{j} [(\omega_i + \neg x^{\mathrm{G},0}) \cdot (\omega_i + \neg x^{\mathrm{G},1})].\end{aligned}$$

Hence, the faulty value of a node x is computed by its original formula provided the good value is specified (i.e., $x^{\mathrm{G},0} + x^{\mathrm{G},1} = 1$). In contrast, if the good value is unspecified (i.e., $x^{\mathrm{G},0} + x^{\mathrm{G},1} = 0$), then the faulty value is *forced* to also be unspecified.

The formulas for $\varphi^{\mathrm{S}}_{u,x}$ and for $\varphi^{\mathrm{F}}_{u,x}$ are defined so that an unspecified good value immediately implies an unspecified faulty value and $x^{\mathrm{S}} = 0$. Thus, propagation

Table V. Definition of the Fault Detection Problem for the Stem Fault z s-a-v

Sub-formula/Condition	Clause Set
Good Circuit	$\varphi_u^{\mathrm{G}} = \bigcup_{x \in V_C} \varphi_{u,x}^{\mathrm{G}}$
Faulty Circuit	$\varphi_u^{\mathrm{F}} = \bigcup_{x \in O^*(z)} \varphi_{u,x}^{\mathrm{F}}$
Node Sensitization	$\varphi_u^{\mathrm{S}} = \bigcup_{x \in O^*(z)} \varphi_{u,x}^{\mathrm{S}}$
Propagation Blocking Conditions	$\varphi_u^{\mathrm{B}} = (\neg x^{\mathrm{S}}) \qquad x \in K_O(z) - O^*(z)$
Fault Activation Conditions	$\varphi_u^{\mathrm{A}} = \begin{cases} (z^{\mathrm{S}}) \cdot (\neg z^{\mathrm{G},1}) \cdot (z^{\mathrm{G},0}) \cdot (z^{\mathrm{F},1}) \cdot (\neg z^{\mathrm{F},0}) & \text{if } v = 1 \\ (z^{\mathrm{S}}) \cdot (z^{\mathrm{G},1}) \cdot (\neg z^{\mathrm{G},0}) \cdot (\neg z^{\mathrm{F},1}) \cdot (z^{\mathrm{F},0}) & \text{if } v = 0 \end{cases}$
Fault Detection Requirement	$\varphi_u^{\mathrm{R}} = \left(\sum_{x \in PO \wedge x \in O^*(z)} x^{\mathrm{S}} \right)$
Detection of Fault z s-a-v	$\varphi_u^{\mathrm{D}} = \varphi_u^{\mathrm{G}} \cup \varphi_u^{\mathrm{F}} \cup \varphi_u^{\mathrm{S}} \cup \varphi_u^{\mathrm{B}} \cup \varphi_u^{\mathrm{A}} \cup \varphi_u^{\mathrm{R}}$

of the error signal is only permitted in the presence of properly specified values for the good circuit variables.

4. COMPUTING MINIMUM SIZE TEST PATTERNS

In this section, we develop the optimization model for computing minimum-size test patterns. This optimization model is based on test pattern generation in the presence of incompletely specified primary input assignments. Moreover, stem faults are assumed throughout, even though the same approach is readily applied to fanout–branch faults.

4.1 The Complete Optimization Model

The main objective of test pattern minimization is to identify the minimum number of primary input assignments that detect the fault. Hence, our goal is to minimize the number of specified primary input assignments such that the given fault is still detected. As a result, we obtain the following optimization model

$$\begin{aligned} \text{minimize} \quad & \sum_{x \in PI} (x^0 + x^1) \\ \text{subject to} \quad & \varphi_u^{\mathrm{D}}, \end{aligned} \tag{13}$$

which basically requires that the total number of assigned input variables be minimized under the constraint that the fault be detected. (Observe that we have $0 \leq x^0 + x^1 \leq 1$ given (6), which implies an upper bound on the value of

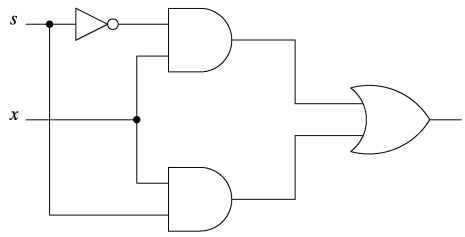


Fig. 3. Minimum-size test pattern for which no propagation path exists.

the cost function of $|PI|$.) Given the mapping between CNF clauses and linear inequalities [Pizzuti 1996] we immediately conclude that (13) corresponds to an integer linear program, and so different integer linear optimization packages can be used for solving the test pattern minimization problem. Nevertheless, the constraints of (13) are tightly related with propositional satisfiability. Consequently, and as shown in Manquinho et al. [1997], SAT-based ILP solvers are preferable for solving ILPs for which the constraints correspond to CNF formulas. For the experimental results given in Section 5, the SAT-based ILP solver of Manquinho et al. [1997] was used.

Furthermore, we note that the optimization model of (13) can be viewed as a formalization of guided pseudo-exhaustive ternary simulation on the primary inputs of a combinational circuit, with the objective of minimizing the number of specified primary inputs assignments, and given the constraint that the fault is detected. The proposed model casts this basic idea into an ILP formulation, thus providing a formal framework for describing the problem and allowing a significant number of algorithms and theoretical results from integer optimization to be used. The validity of the proposed optimization model is formally establish in Flores et al. [2000].

4.2 Limitations of the Model

In general, there may exist faults for which it is possible to identify test patterns with a smaller number of specified assignments, but which do not uniquely identify a set of sensitizable paths. Let us consider the example circuit in Figure 3. Let the target fault be x s-a-1. From the circuit, it is clear that any assignment to the selection variables s permits detecting the fault. Hence, a valid test pattern is $T = \{(X, 0)\}$, since any assignment to the remaining variable permits detecting the fault. However, observe that T by itself does not yield any sensitization path for the fault to be detected. Only the additional assignment to the remaining primary input allows the fault effect to propagate to the primary outputs. Consequently, any test generation model based on the D-calculus [Abramovici et al. 1990] or any of its derivations is by itself unable to identify such test patterns, since for some cases propagation does not actually take place and only the propagation conditions are implicitly validated. As a result, our proposed model yields the minimum-size test patterns that guarantee, given the specified assignments, propagation of the fault effect to a primary output by *explicitly* defining one or more sensitizable paths.

Table VI. Experimental Results for the IWLS'89 Benchmarks (Allowing 1000 Conflicts per Faults)

				ATALANTA			MTP					
Circuit	#PI	#G	#F	#R	#A	%X	#R	#A	%X	Δ	%Opt	time/ fault
9symml	9	157	752	2	0	1.4	2	0	8.9	7.5	100	2.04
cht	47	209	820	0	0	93.6	0	0	94.4	0/8	100	0.64
cm138a	6	26	124	0	0	16.7	0	0	16.7	0.0	100	0.02
cm150a	21	62	232	0	0	68.4	0	0	71.0	2.6	100	1.55
cm163a	16	54	220	0	0	70.7	0	0	72.8	2.1	100	0.28
cmb	16	54	248	0	0	29.6	0	0	30.0	0.4	100	0.07
comp	32	105	480	1	0	24.0	1	0	39.6	15.6	2	10.64
comp16	35	221	960	0	0	30.7	0	0	32.9	2.2	4	13.66
cordic	23	74	342	0	0	30.7	0	0	40.2	9.5	37	6.28
cu	14	51	262	7	0	53.0	7	0	57.1	4.1	100	0.14
majority	5	12	54	0	0	8.5	0	0	8.5	0.0	100	0.01
misex1	8	52	224	0	0	49.8	0	0	54.4	4.6	100	0.17
misex2	25	84	422	0	0	73.5	0	0	75.8	2.3	100	0.20
misex3	14	533	2590	7	0	24.4	7	0	37.7	13.3	76	25.29
mux	21	47	202	0	0	67.3	0	0	75.8	8.5	100	0.94
pcle	19	76	328	0	0	73.3	0	0	74.9	1.6	99	0.45
pcler8	27	94	400	0	0	78.1	0	0	79.2	1.1	98	1.97
term1	34	155	708	6	0	72.2	6	0	74.42	2.2	86	4.35
too_large	38	234	1132	15	0	54.9	15	0	62.2	7.3	20	18.27
unreg	36	103	448	0	0	90.6	0	0	91.7	1.1	86	0.93

5. EXPERIMENTAL RESULTS

The model described in the previous section has been integrated in a test pattern generation framework for the computation of minimum size test patterns referred to as *Minimum Test Pattern generator* (MTP), which uses the SAT-based ILP algorithm *bsolo* [Manquinho et al. 1997] and the fault simulator provided with ATALANTA [Lee and Ha 1993]. The results included below were obtained with the IWLS'89 benchmark suite [IWLS 1989] and with the ISCAS'85 benchmark suite [Brglez and Fujiwara 1985]. In all cases, MTP was run with a bound on the amount of allowed search (i.e., the total number of conflicts). This permits MTP to identify acceptable solutions, which in some cases may not be necessarily optimal. Moreover, in order to speed up convergence to the optimal solutions, MTP can use the solution computed by ATALANTA (or by any other ATPG tool) as the startup assignment. These assignments provide an initial upper bound on the value of the optimal solution. If ATALANTA aborts the fault, then TG-GRASP [Silva and Sakallah 1997] is used for computing a startup test pattern.

Table VI contains the results for the IWLS'89 benchmarks for both ATALANTA and MTP. ATALANTA is an ATPG tool that can generate test patterns with don't cares. For each benchmark *all* faults were targeted in order to allow for a meaningful comparison between the two algorithms. Columns #PI, #G, #F, #R and #A denote, respectively, the number of primary inputs, gates, faults, redundant faults and aborted faults. %X denotes the percentage of don't care bits on all test patterns; Δ denotes the variation in percentage from

Table VII. Experimental Results for the ISCAS'85 Benchmarks (Allowing 100 Conflicts per Fault)

				ATALANTA			MTP					
Circuit	#PI	#G	#F	#R	#A	%X	#R	#A	%X	Δ	%Opt	time/ fault
c432	36	160	524	3	1	56.2	4	0	60.8	4.6	0	3.21
c499	41	202	758	8	0	17.1	8	0	18.7	1.6	0	4.35
c880	60	383	942	0	0	82.2	0	0	83.8	1.6	12	2.54
c1355	41	546	1574	8	0	13.3	8	0	13.7	0.4	0	9.12
c1908	33	880	1878	8	0	44.7	8	0	48.4	3.7	0	9.61
c2670	233	1193	2746	97	20	92.0	117	0	92.4	0.4	23	10.99
c3540	50	1669	3425	134	0	74.6	134	0	77.3	2.7	15	16.81
c5315	178	2307	5350	59	0	92.6	59	0	92.9	0.3	14	9.34
c6288	32	2416	7744	34	387	22.2	34	0	25.1	2.9	1	36.65
c7552	207	3512	7550	77	181	86.9	131	0	86.9	0/0	4	17.46

ATALANTA to *MTP*; %Opt denotes the percentage of faults for which *MTP* was able to find the actual minimum-size test pattern. Finally, time/fault denotes the average time in seconds spent solving the *ILP* for each fault.[4]

From these results several conclusions can be drawn. First, MTP allows validating the heuristics used in ATALANTA for computing test patterns with "don't cares." Indeed for several benchmarks, ATALANTA already identifies the minimum-size test patterns for all faults. Nevertheless, for other benchmarks, the test patterns computed by ATALANTA can be far from the minimum-size test patterns. For these cases the percentage of "don't cares" computed with MTP can be as much as 15% above the values computed by ATALANTA. Finally, we observe that for medium-size circuits MTP is able to compute the actual minimum-size test patterns for all faults in the circuit in a reasonable amount of time per fault. For larger circuits, MTP finds solutions that are better than those computed by ATALANTA, but which are not guaranteed to be optimal.

Table VII contains the results for the ISCAS'85 circuits.[5] For these benchmarks a smaller search effort (i.e., 100 conflicts) was allowed. This leads to smaller run times and, consequently, less optimal results. Once more we can conclude that MTP is able to improve over the ATALANTA results, but in this case the improvements are in general smaller, since it becomes harder for the ILP solver *bsolo* to find optimal solutions. (As can be concluded the percentage of optimal solutions found ranges from 0 to 20 percent.) For some of these circuits, we run MTP with a larger number of allowed conflicts (i.e., 1000 conflicts). The obtained results are shown in Table VIII. As can be observed, a larger percentage of unspecified input assignments is obtained at the cost of a larger search effort per fault. Accordingly, the time per fault also increases.

From the previous experimental results for the IWLS'89 and ISCAS'85 benchmarks, we can draw the following conclusions:

[4]Note that even though the time per fault for ATALANTA is not shown, the average time observed was 0.1 second for the benchmarks considered.

[5]Observe that ATALANTA aborts some faults for several benchmarks. For those cases, MTP uses TG-GRASP as the start-up ATPG tool, and consequently does not abort any fault.

Table VIII. Experimental Results for Some of the ISCAS'85 Benchmarks (Allowing 1000 Conflicts per Fault)

				ATALANTA			MTP					
Circuit	#PI	#G	#F	#R	#A	%X	#R	#A	%X	Δ	%Opt	time/ fault
c432	36	160	524	3	1	56.2	4	0	64.1	7.9	2	27.04
c499	41	202	758	8	0	17.1	8	0	19.5	2.4	0	33.71
c880	60	383	942	0	0	82.2	0	0	85.6	3.4	40	22.34
c1355	41	546	1574	8	0	13.3	8	0	15.2	1.9	0	64.86
c1908	33	880	1878	8	0	44.7	8	0	60.0	15.3	1	73.44
c2670	233	1193	2746	97	20	92.0	117	0	93.0	1.0	25	83.46

—For some circuits the heuristics used by ATALANTA, as well as by other structural ATPG algorithms, are extremely effective and MTP can be used to formally prove this result.

—Whenever the main goal is maximizing the number of "don't care" bits, then MTP can be run on top of ATALANTA (or any other ATPG algorithm); thus, in general allowing for an increased number of unspecified bit assignments. The improvements obtained by MTP are related to the amount of allowed search effort, and MTP is always guaranteed to produce results that are no worse than the startup tool (in our case ATALANTA or TG-GRASP).

6. CONCLUSIONS

In this article, we introduce a SAT-based integer linear programming model for computing minimum-size test patterns. The applicability of the model has been illustrated by computing minimum size test patterns for several benchmark circuits. The next step of this work is to study the application of minimum-size test patterns to the synthesis of BIST logic, with the objective of evaluating the reduction in size of the synthesized logic obtained by using MTP.

Additional research work involves further constraining the ILP formulation so that larger problem instances can be solved optimally. Furthermore, the trade-offs between minimum-size test pattern computation, fault simulation and fault compaction need to be studied. Finally, a long-term objective of this work is the integration of the proposed model in a complete testing environment, thus enabling the use of minimum-size test patterns for different purposes, such as the validation of test pattern minimization heuristics or the synthesis of reduced-size FSMs for BIST in specific target applications.

REFERENCES

ABRAMOVICI, M., BREUER, M. A., AND FRIEDMAN, A. D. 1990. *Digital Systems Testing and Testable Design*. IEEE Press, New York.

BRGLEZ, F., AND FUJIWARA, H. 1985. A neutral list of 10 combinational benchmark circuits and a target translator in FORTRAN. In *Proceedings of the IEEE International Symposium on Circuits and Systems*.

CHAKRABARTY, K., MURRAY, B. T., LIU, J., AND ZHU, M. 1997. Test width compression for built-in self testing. In *Proceedings of the International Test Conference* (Nov.).

CHAKRADHAR, S. T., AGRAWAL, V. D., AND ROTHWEILER, S. G. 1993. A transitive closure algorithm for test generation. *IEEE Trans. Comput.-Aided Des. 12*, 7 (July), 1015–1028.

COX, H., AND RAJSKI, J. 1994. On necessary and nonconflicting assignments in algorithmic test patterns generation. *IEEE Trans. Comput.-Aided Des. 13*, 4 (Apr.), 515–530.

FLORES, P., NETO, H., AND MARQUES-SILVA, J. 2000. On computing minimum size test patterns. Tech. Rep. RT/008/2000 (Sept.). INESC, Lisbon, Portugal.

FUJIWARA, H., AND SHIMONO, T. 1983. On the acceleration of test generation algorithms. *IEEE Trans. Comput. 32*, 12 (Dec.), 1137–1144.

GIRALDI, J., AND BUSHNELL, M. L. 1991. Search state equivalence for redundancy identification and test generation. In *Proceedings of the International Test Conference*, pp. 184–193.

GOEL, P. 1981. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Trans. Comput. 30*, 3 (Mar.), 215–222.

HELLEBRAND, S., REEB, B., TARNICK, S., AND WUNDERLICH, H.-J. 1995. Pattern generation for deterministic BIST scheme. In *Proceedings of the International Conference on Computer-Aided Design*.

IWLS. 1989. Test benchmark suite. *International Workshop on Logic Synthesis 1989*. Available from `http://www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth89/`.

KIRKLAND, T., AND MERCER, M. R. 1987. A topological search algorithm for atpg. In *Proceedings of the Design Automation Conference*, pp. 502–508.

KUNZ, W., AND PRADHAM, D. K. 1992. Recursive learning: An attractive alternative to the decision tree for test generation in digital circuits. In *Proceedings of the International Test Conference*, pp. 816–825.

LARRABEE, T. 1992. Test pattern generation using Boolean satisfiability. *IEEE Trans. Comput.-Aided Des. 11*, 1 (Jan.), 4–15.

LEE, H. K., AND HA, D. S. 1993. On the generation of test patterns for combinational circuits. Tech. Rep. 12_93, Department of Electrical Engineering, Virginia Polytechnic Institute and State University.

MANQUINHO, V., FLORES, P., MARQUES-SILVA, J., AND OLIVEIRA, A. L. 1997. Prime implicant computation using satisfiability algorithms. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence* (Nov.). IEEE Computer Society Press, Los Alamitos, Calif.

NIERMANN, T. M., AND PATEL, J. H. 1991. HITEC: A test generation package for sequential circuits. In *Proceedings of European Conference on Design Automation* (Feb.).

PIZZUTI, C. 1996. Computing prime implicants by integer programming. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence* (Nov.). IEEE Computer Society Press, Los Alamitos, Calif.

POMERANZ, I., REDDY, L. N., AND REDDY, S. M. 1993. COMPACTEST: A method to generate compact test sets for combinational circuits. *IEEE Trans. Comput.-Aided Des. Integ. Circ. Syst. 12*, 7 (July), 1040–1049.

SCHULZ, M. H., AND AUTH, E. 1989. An improved deterministic test pattern generation with applications to redundancy identification. *IEEE Trans. Comput.-Aided Des. 8*, 7 (July), 811–816.

SILVA, J. P. M., AND SAKALLAH, K. A. 1994. Dynamic search-space pruning techniques in path sensitization. In *Proceedings of the Design Automation Conference*, pp. 705–711.

SILVA, J. P. M., AND SAKALLAH, K. A. 1997. Robust search algorithms for test pattern generation,. In *Proceedings of the Fault-Tolerant Computing Symposium* (June 1997).

STEPHAN, P. R., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. L. 1996. Combinational test generation using satisfiability. *IEEE Trans. Comput.-Aided Des. 15*, 9 (Sept.), 1167–1176.

TERAMOTO, M. 1993. A method for reducing the search space in test pattern generation. In *Proceedings of the International Test Conference*, pp. 429–435.

Received February 1999; accepted June 2001