An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits

PRABHAKAR GOEL

Abstract—**The D-algorithm (DALG) is shown to be ineffective for the class of combinational logic circuits that is used to implement error correction and translation (ECAT) functions. PODEM (path-oriented decision making) is a new test generation algorithm for combinational logic circuits. PODEM uses an implicit enumeration approach analogous to that used for solving 0–1 integer programming problems. It is shown that PODEM is very efficient for ECAT circuits and is significantly more efficient than DALG over the general spectrum of combinational logic circuits. A distinctive feature of PODEM is its simplicity when compared to the D-algorithm. PODEM is a complete algorithm in that it will generate a test if one exists. Heuristics are used to achieve an efficient implicit search of the space of all possible primary input patterns until either a test is found or the space is exhausted.**

Index Terms—**Combinational logic, D-algorithm, decision tree, error correction, implicit enumeration, stuck faults, test generation, untestable fault.**

INTRODUCTION

TEST generation for combinational circuits has traditionally been considered a solved problem [1]. However, evaluation studies conducted by us have shown that the D-algorithm (DALG) [2] is extremely inefficient in generating tests for combinational logic circuits that implement error correction and translation (ECAT) type functions. For practical purposes, a logic circuit is characterized as being ECAT type if portions of it are constituted by XOR (EXCLUSIVE-OR) gates with reconvergent fan-out. The increasing emphasis on reliability of computer systems resulted in increased usage of ECAT circuits and existing test generation tools proved to be inadequate for them. An implicit enumeration [3] algorithm (PODEM—path oriented decision making) was developed shortly thereafter and experiments demonstrate that PODEM is an effective test generator for ECAT circuits. Subsequent studies, which are reported in this paper, show that the PODEM algorithm is significantly faster than DALG over the general class of combinational logic circuits. For LSI logic circuits there is an increasing trend towards designing for testability using the level sensitive scan design approach [9] or equivalent approaches [10]. Level sensitive scan design reduces the test generation problem for sequential logic circuits to one for combinational logic circuits. Advances in the state of the art for combinational logic test generation have therefore assumed increased significance.

Manuscript received November 23, 1979; revised October 6, 1980.

The author is with IBM General Technology Division, Poughkeepsie, NY 12601.

The unique characteristics of ECAT circuits that caused DALG to perform poorly are discussed in this paper. The PODEM test generation algorithm for combinational circuits is described. Analogies are drawn between PODEM and the implicit enumeration algorithms used in integer programming. Quantitative data is provided to demonstrate the performance advantage of PODEM over DALG. As is true for DALG, PODEM is a complete algorithm in that it will generate a test for a stuck fault if a test exists.

DEFINITIONS

Common terminology pertaining to test generation for logic circuits is readily introduced with an example. Fig. 1 shows a combinational logic circuit and a test for the single stuck fault that causes net h to permanently assume a 0 state. A *stuck-at-1 (s-a-1) fault* on a signal net causes that net to permanently assume the 1 state. A *stuck-at-0 (s-a-0) fault* causes a permanent 0 on the faulted net. The five-valued logic [2] (0, 1, X, D, $\overline{D}$) of Table I is used to describe the behavior of a circuit with failures. The *value D* designates a logic value 1 for a net in the good circuit or good machine and a 0 for the same net in the failing machine, $\overline{D}$ is the complement of D, and X designates a DON'T CARE value. A behavior difference between the good circuit and the failing circuit propagates along a *sensitized path*. In Fig. 1 the signal path h, j is referred to as a sensitized path. Externally controllable nets are referred to as *primary inputs* and are abbreviated to PI's. Externally observable nets are referred to as *primary outputs* and are abbreviated to PO's. Each stuck fault is associated with either an input net or the output net of a unique gate which is referred to as the *gate under test* and abbreviated to G.U.T. A *test* is generated when a sensitized path is built from the output of the gate under test to some PO. In Fig. 1 assignment of the values, X, X, 1, 1, 0 to the PI nets a, b, c, d, e, respectively, along with a measurement on the PO net j constitutes a test for the fault h s-a-0. The gate driving net h is the gate under test for the fault h s-a-0.

THE ECAT PROBLEM

The first ECAT type circuit encountered was an implementation of a single error correcting/double error detecting scheme over a set of 64 data bits and 8 parity bits. Error detection schemes involve the use of EXCLUSIVE-OR (XOR) trees to determine parities over various combinations of data/parity bits. The resulting parity signals are further combined to

0018-9340/81/0300-0215$00.75

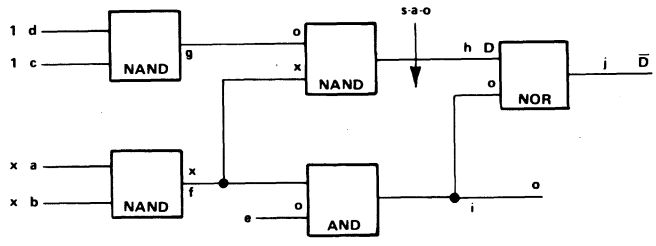

Fig. 1. Example to illustrate test generator terminology.

TABLE I
FIVE-VALUED LOGIC SYSTEM USED IN TEST GENERATION

B= NOT.A

A	B
0	1
1	0
x	x
D	D̄
D̄	D

C=A.AND.B

A \ B	0	1	x	D	D̄
0	0	0	0	0	0
1	0	1	x	D	D̄
x	0	x	x	x	x
D	0	D	x	D	0
D̄	0	D̄	x	0	D̄

XOR = EXCLUSIVE OR GATE
$\overline{\text{XOR}}$ = EQUIVALENCE GATE

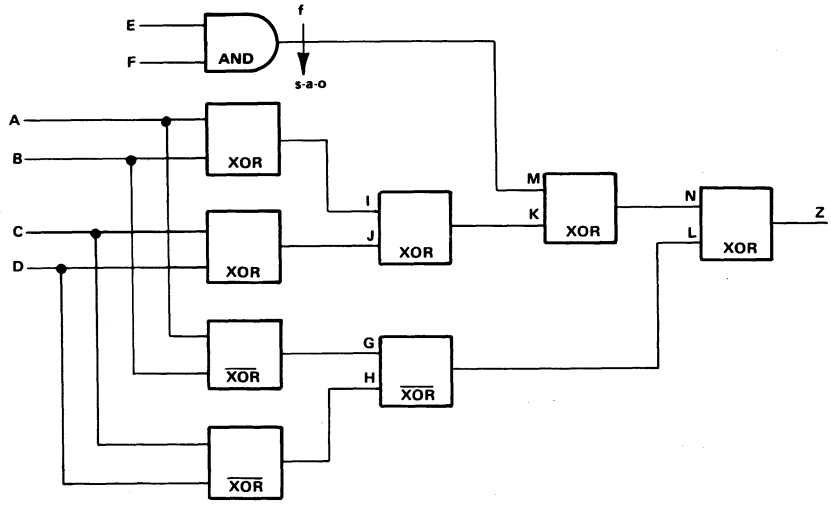

Fig. 2. ECAT type circuit exhibiting fanout and reconvergence involving XOR trees. Depicted fault illustrates DALG's inefficiency for such circuits.

generate signals that indicate error or help correct error bits. Basically, the ECAT circuit could be viewed as being made up of some number of XOR trees with reconvergence of the generated parity signals. A circuit with the basic ECAT characteristics is referred to as an "ECAT type circuit" or an "ECAT circuit." The circuit in Fig. 2 is an ECAT circuit.

DALG is typically used with the primitive gate (AND, OR, NAND, NOR) representation of a logic circuit. However, for ease of explanation DALG's techniques are extended to the XOR and equivalence gates. In generating a test, DALG creates a decision structure in which there is more than one choice available at each decision node. Through an implicit enumeration process, all alternatives at each decision node are capable of being examined. For the fault f in Fig. 2, DALG may typically go through the following steps.

1) The test for f requires a logic 1 on M for the good machine. Setting E and F each to 1 results in a D at M.

2) Generating a sensitized path from the net M to the primary output Z, using recursive intersection of D-cubes, may result in the ordered assignments $K = 1$ and $L = 1$ represented in Fig. 3. Alternative assignments $K = 0$ and $L = 0$ are still available for consideration should the present assignments prove futile.

3) DALG justifies each internal net assignment on a levelized basis (see Fig. 3). Since the functions P and $\overline{P}$ realized at nets K and L, respectively, are complementary, no justification is possible for the concurrent assignments $K = 1$, $L = 1$. However, in establishing the absence of the justification, DALG must enumerate 2^3 primary input values (2^n if $2n$ were the number of external inputs to the XOR tree) before it can correct the bad decision made on L, that is, change the assignment on L from 1 to 0. Looked at differently, DALG made two conflicting assignments $K = 1$ and $L = 1$ (on internal nets) and to detect the conflict DALG is forced to enumerate half of all possible values on primary inputs A–D. The requirement of having all external inputs known to make the output known is peculiar to XOR (or equivalence) trees.

Next consider what would happen for parity trees with up to 72 external inputs (as in the original ECAT circuit). For some faults the number of primary input values that must be enumerated can be of the order of 2^{36}. With the above phenomenon repeating itself for most faults in ECAT circuits, it is seen that even if DALG were implemented on the fastest computers, it would have unacceptable performance in run time and therefore in test coverage.

An experimental study conducted by us demonstrated that DALG was ineffective for ECAT type circuits. When permitted to give up on faults after a large number of remade decisions, DALG was able to generate tests for only six out of 7000 faults in 2 h of CPU time with an IBM 360/85 computer and DALG had still not completed.

TEST GENERATION VIEWED AS A FINITE SPACE SEARCH PROBLEM

The test generation problem can be formulated as a search of the n-dimensional 0–1 state space of primary input patterns of an n-input combinational logic circuit. Consider the combinational circuit of Fig. 4. In Fig. 4 g is an internal net and

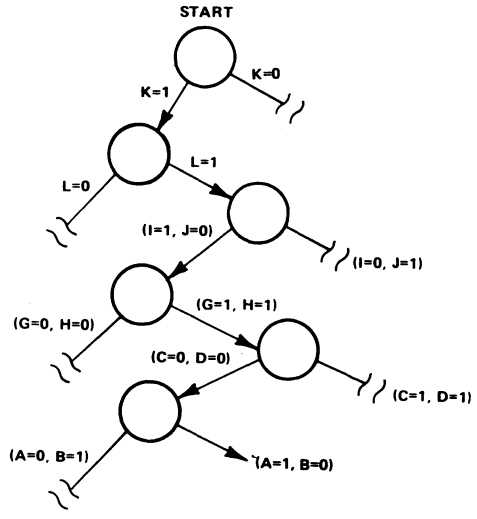

Fig. 3. DALG decision tree—DALG needs a large number of remade decisions resulting in a high test generation cost.

the objective is to generate a test for the stuck fault g s-a-0. The state on g can be expressed as a Boolean function of the primary inputs, $X_1, X_2, \cdots, X_n$. Similarly, each primary output ($f_j, j = 1, 2, \cdots, m$) can be expressed as a Boolean function of the state on net g as well as the primary inputs $X_1, X_2, \cdots, X_n$. Let

$$g = G(X_1, X_2, \cdots, X_n)$$

and

$$f_j = F_j(g, X_1, X_2, \cdots, X_n)$$

where $1 \leq j \leq m$ and $X_i = 0$ or 1 for $1 \leq i \leq n$.

The problem of test generation for g s-a-0 can be stated [4] as one of solving the following set of Boolean equations:

$$G(X_1, X_2, \cdots, X_n) = 1$$

$$F_j(1, X_1, X_2, \cdots, X_n) \oplus F_j(0, X_1, X_2, \cdots, X_n) = 1$$

for at least one j, $1 \leq j \leq m$ and $X_i = 0$ or 1 for $1 \leq i \leq n$.

The set of equations for g s-a-1 are the same as above except that G is set equal to 0. Hence, test generation can be viewed as a search of an n-dimensional 0–1 space defined by the variables X_i ($1 \leq i \leq n$) for points that satisfy the above set of equations. More generally, the search will result in finding a k-dimensional subspace ($k \leq n$) such that all points in the subspace will satisfy the above set of equations.

The integer programming problem and various problems in the field of artifical intelligence have been approached using state space search methods [5] and the branch and bound technique [6]. Implicit enumeration refers to a subset of the branch and bound algorithms designed specifically for search of a n-dimensional 0–1 state space.

PODEM Test Generation Algorithm

The PODEM (path oriented decision making) test generation algorithm is an implicit enumeration algorithm in which all possible primary input patterns are implicitly, but exhaustively, examined as tests for a given fault. The examination

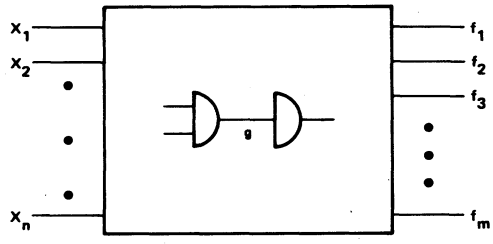

Fig. 4. Combinational circuit used in formulating test generation as an n-dimensional 0–1 state space search problem.

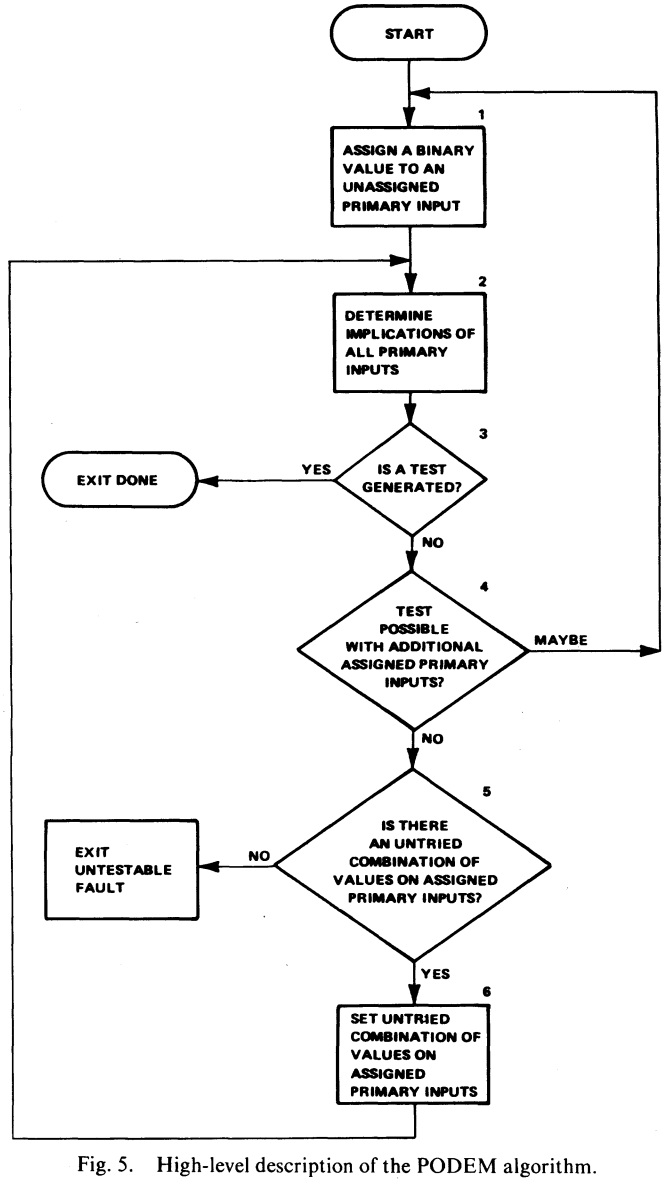

Fig. 5. High-level description of the PODEM algorithm.

of PI patterns is terminated as soon as a test is found. If it is determined that no PI pattern can be a test, the fault is *untestable* by definition.

Boolean equations were used in the preceding section for conceptual purposes. Unlike the Boolean difference approach [4], the PODEM algorithm does not involve manipulation of Boolean equations. Instead, the combinational logic schematic is used in a manner similar to that in the D-algorithm. The flowchart of Fig. 5 provides a high-level description of the PODEM algorithm. The implicit enumeration process used in PODEM results in the decision tree structure illustrated by

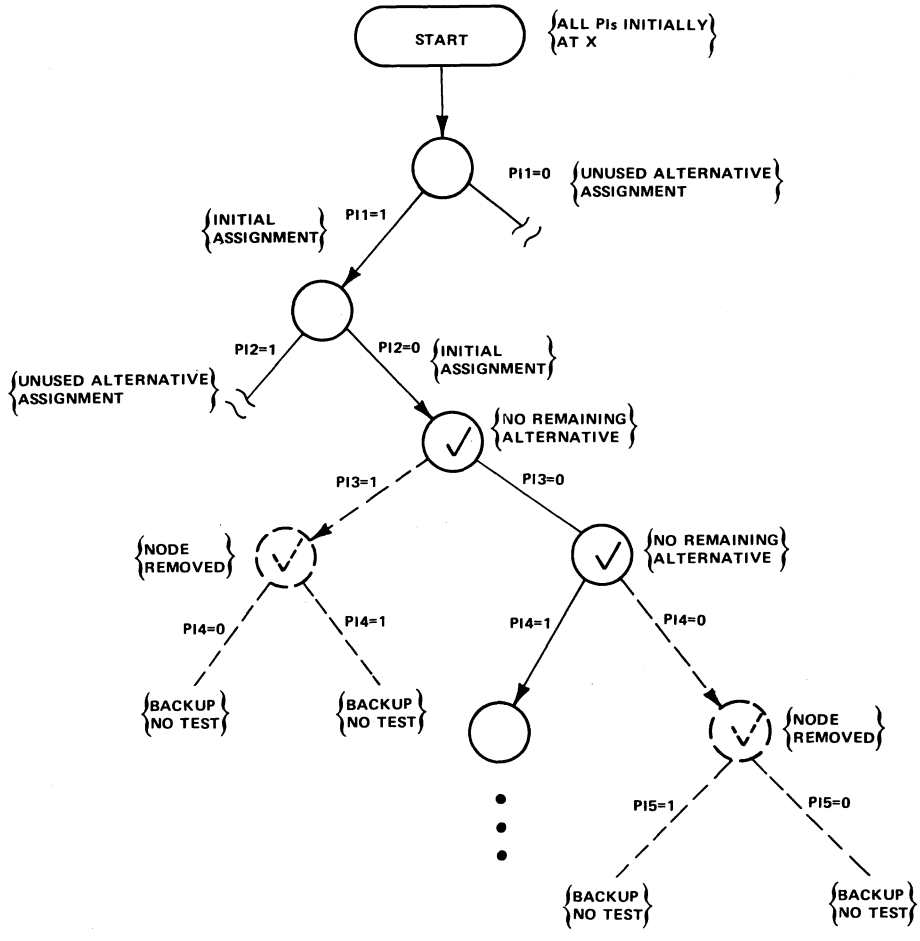

Fig. 6. Decision tree illustrating the "branch and bound" process in the PODEM algorithm.

Fig. 6. In Figs. 5 and 6 all PI's are initially at X, that is, they are unassigned. An initial assignment ("branch"–in the context of branch and bound algorithms) of either 0 or 1 on a PI is recorded as an unflagged node in the decision tree. Implications of present PI assignments (box 2) use the five-valued logic of Table I and the process is identical to the forward IMPLY process in the D-algorithm. The decision tree is an ordered list of nodes with: 1) each node identifying a current assignment of either a 0 or 1 to one PI, and 2) the ordering reflects the relative sequence in which the current assignments were made. A node is flagged (indicated by a check mark inside the node in Fig. 6) if the initial assignment has been rejected and the alternative is being tried. When both assignment choices at a node are rejected, then the associated node is removed and the predecessor node's current assignment is also rejected. The last PI assignment made is rejected if it can be determined (box 4 of Fig. 5) that no test can be generated with the assignments made on the assigned PI's, regardless of values that may be assigned to the as yet unassigned PI's. The rejection of a PI assignment results in a "bounding" of the decision tree, in the context of branch and bound algorithms, since it avoids the enumeration of the subsequent assignments to the as yet unassigned PI's. The two simple propositions listed below are evaluated to carry out the process of box 4 in Fig. 5.

Proposition 1: The signal net (for the given stuck fault) has the same logic level as the stuck level.

Proposition 2: There is no signal path from an internal signal net to a primary output such that the internal signal net is at a D or $\bar{D}$ value and all other nets on the signal path are at X.

If Proposition 1 is true, then it is obvious that assignment of known values to unassigned PI's cannot result in a test. If Proposition 2 is true, then there exists no signal path along which the D or $\bar{D}$ can propagate to a circuit output. Hence, only if both Propositions 1 and 2 are false, then in box 4 it is concluded that a test is possible with the present PI assignment even though further enumeration may show that it is not.

Fig. 7 is a more detailed description of the basic PODEM algorithm and provides the steps used to carry out the "bounding" of the decision tree—see boxes 5 through 9. The decision tree being an ordered list of nodes can be implemented as a LIFO stack. An initial PI assignment (box 2) results in pushing the associated unflagged node onto the stack. The "bounding" of the decision tree is done by popping the stack until an unflagged node is on the top of the stack. The alternative assignment is made on the PI associated with the unflagged node and the node is modified to be flagged (box 8). All nodes popped out of the stack are in effect removed from

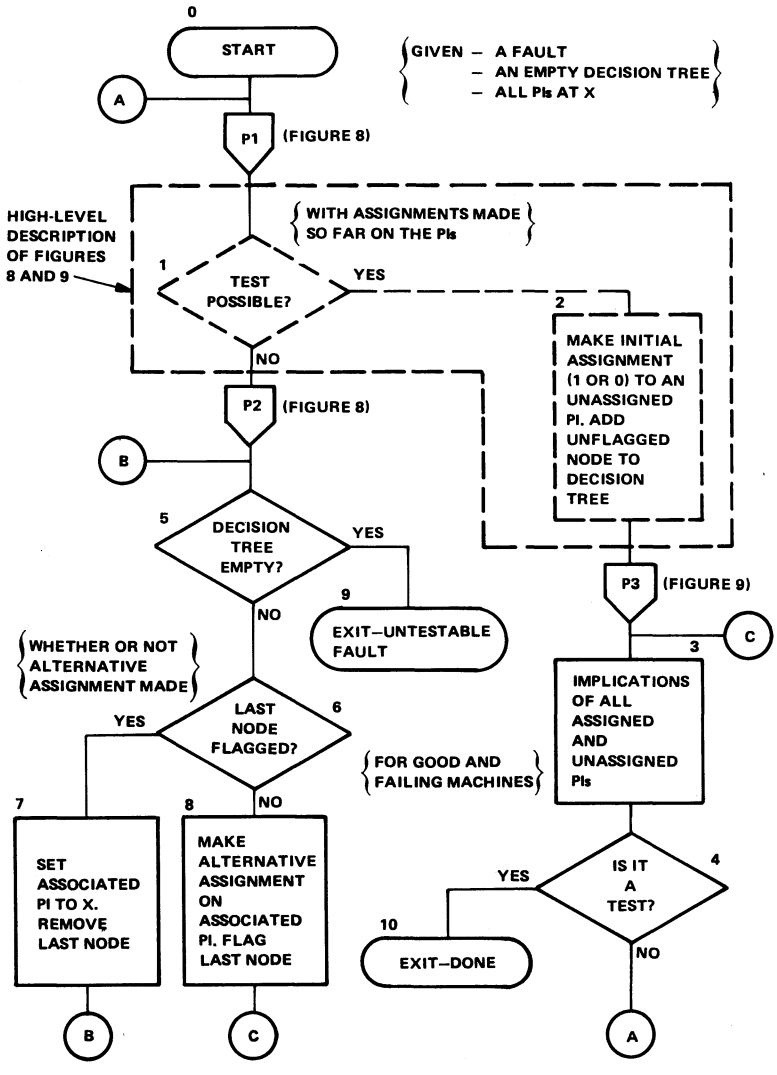

Fig. 7. Basic PODEM algorithm detailing the bounding of the decision tree.

the decision tree and their associated PI's are set to X (box 7). A new combination of values on the PI's is obtained by the bounding process and is evaluated (boxes 3 and 4) for constituting a test. The entire process is iteratively continued until a test is found (box 10) or it is determined that a test is possible with additional PI assignments (box 2) or the decision tree becomes empty (box 9). Note that bounding of the decision tree results in rejection of only those combinations of values on the PI's that cannot be a test for the given stuck fault. Therefore, if the decision tree becomes empty the given fault must be untestable. It is thus shown that PODEM is a complete test generation algorithm in that a test will be found if one exists.

In the flowchart of Fig. 7 it would be most desirable to have the least number of remade decisions (reversals of initial PI assignments) before either a test is found or a fault is determined to be untestable. Practical considerations dictate that when a preset number of decisions are remade, the fault being worked on is given up and a new fault is selected for test generation. Poor initial PI assignments can cause a large number of remade decisions before a test is found. However, the process of determining initial assignments should not be so expensive as to outweigh the benefits of good initial assignments.

In PODEM heuristics are used to select the next PI to be assigned as well as the initial logic level to be assigned (box 2 of Fig. 7). Branch and bound algorithms also use heuristics for the analogous process of "choosing the partitioning variable and branching" [7].

Choosing a Primary Input and Logic Level for Initial Assignment

A two-step process is used to choose a PI and logic level for initial assignment.

Step 1: Determine an initial objective—the objective should be to bring the test generator closer to its goal of propagating a D or $\overline{D}$ to a primary output.

Step 2: Given the initial objective, choose a PI and a logic level such that the chosen logic level assigned to the chosen PI has a good likelihood of helping towards meeting the initial objective.

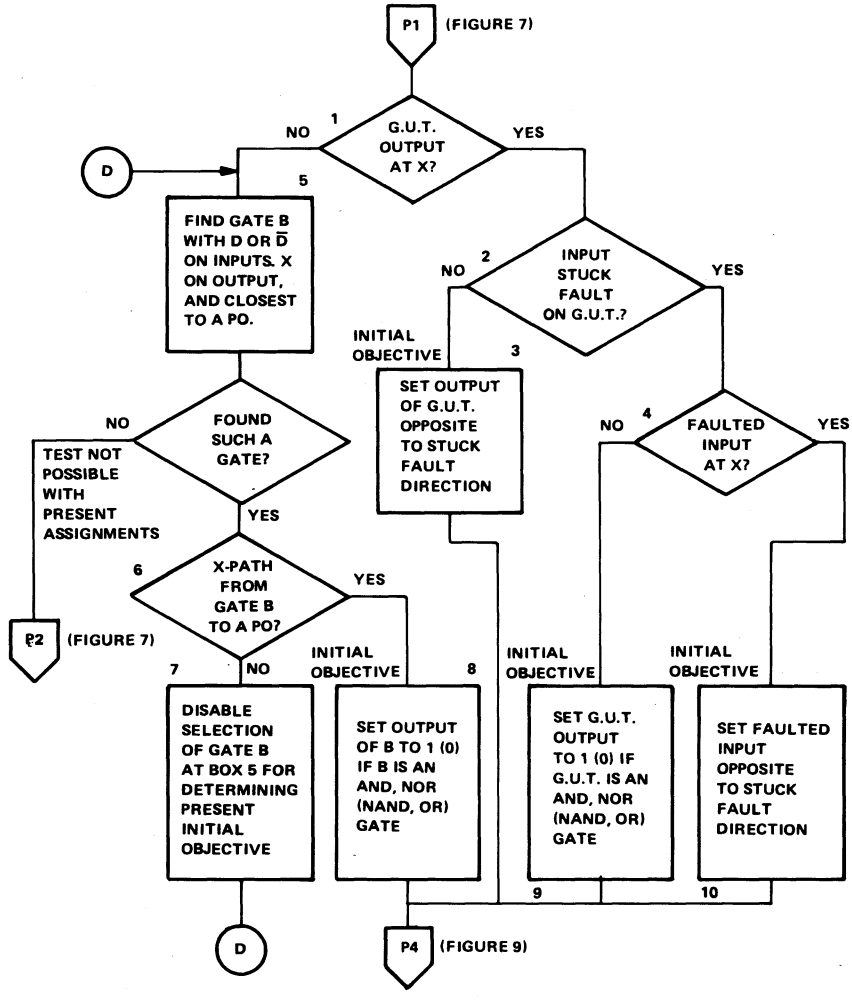

Fig. 8. Determination of initial objective—used as a guide to choose next PI assignment.

Initial objectives are determined through the procedure flowcharted in Fig. 8. An objective is defined by: 1) a logic level 0 or 1 that is referred to as the *objective logic level,* and 2) an *objective net* which is the net at which the objective logic level is desired. If the failing gate or gate under test (G.U.T.) does not have a D or $\overline{D}$ on its output (that is, is not setup), the initial objective is directed towards promoting setup for that gate. Once the gate under test has been setup, the initial objective is aimed at propagating a D or $\overline{D}$ one level of logic closer to a PO than before. In box 5 of Fig. 8 the shortest distance from a gate to a PO is used as a crude measure of the difficulty of creating a sensitized path between the gate and the PO. More sophisticated measures are certainly available although the cost of determining them may exceed the added benefits obtained. In boxes 8 and 9 the statement of the objective is intended only to establish the requirement for sensitizing logic levels on those inputs of gate B which are presently at X.

The flowchart of Fig. 9 describes a procedure, referred to as *backtrace,* to obtain an initial PI assignment given an initial objective. Boxes 4 and 5 of Fig. 9 require relative measures of controllability for the inputs of the logic gate Q. The rationale for choosing the "easiest to control" input or the "hardest to control" input of Q is apparent from the following:

1) In box 5 the current objective level is such that it can be obtained by setting any one input of gate Q to a *controlling state* (e.g., 0 for an AND/NAND gate or 1 for an OR/NOR gate). The intent is to choose that input which can be most easily set.

2) In box 4 the current objective level can only be obtained by setting each input of gate Q to a *noncontrolling state* (e.g., 1 for an AND/NAND gate or 0 for an OR/NOR gate). The intent is to choose that input which is hardest to set, since an early determination of the inability to set the chosen input will prevent fruitless time spent in attempting to set the remaining inputs of Q.

The backtrace procedure causes a path to be traced from the objective net backwards to a primary input. Since the initial PI assignment corresponds to making a decision at a node of the decision tree, the algorithm has been named path-oriented decision making (PODEM) test pattern generator.

The controllability measures for the inputs of Q influence only the efficiency of determining a valid PI assignment that satisfies the chosen initial objective. Hence, heuristic measures for controllability are adequate. Methods for determining heuristic controllability measures are available in published literature [8]. The backtrace process in PODEM is much more simplified than (though similar to) the corresponding *consistency operation* [2] carried out in other test generators.

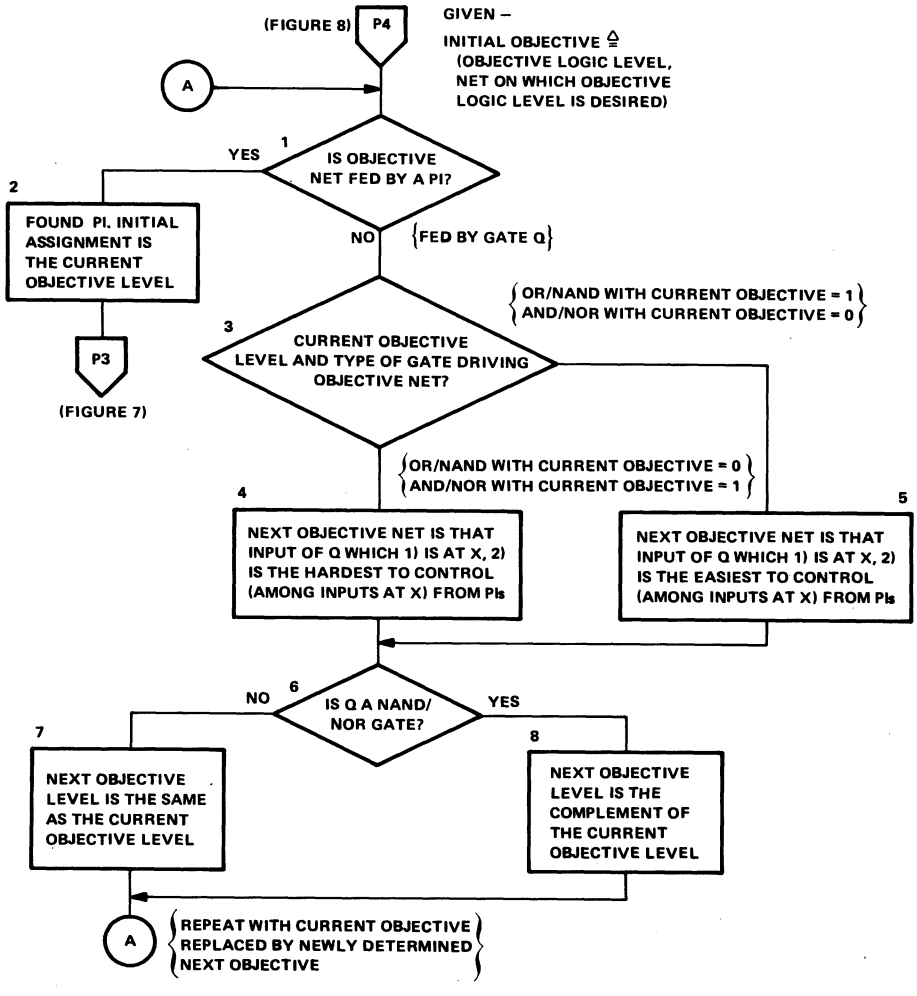

Fig. 9. Backtrace—determine initial PI assignment given an initial objective.

TABLE II
COMPARISON OF PODEM AND DALG

TEST CASE	TYPE	# OF BLOCKS	NORMALIZED RUN TIME		TEST COVERAGE %	
			PODEM	DALG	PODEM	DALG
1	ECAT	828	1	34.5	100.0	99.7
2	ECAT	935	1	12.8	100.0	93.1
3	ECAT	2,002	1	Not meaningful*	100.0	31.2
4	ECAT	948	1	5.7	100.0	95.7
5	-	951	1	2.2	99.5	99.5
6	-	1,249	1	3.5	98.2	98.2
7	-	1,172	1	2.6	98.5	98.5
8	ALU	1,095	1	15.3	96.5	96.2
9	MUX	1,082	1	3.2	96.6	96.6
10	-	915	1	3.9	96.3	96.3
11	ALU	919	1	1.7	99.7	99.8
12	DECODER	1,018	1	3.8	99.1	99.1
13	PLA	538	1	2.5	94.5	94.5
14	PLA	682	2.6	1	89.4	89.4
15	PLA	1,566	1	3.1	97.4	97.4

* Test coverage from DALG was much lower making a run time comparison lose its significance.

PERFORMANCE COMPARISON

A practical version of DALG was compared with an implementation of PODEM. It showed that PODEM achieved 100 percent test coverage on each of the four ECAT type circuits used. Since DALG and PODEM are complete algorithms, given enough time, both will generate tests for each testable fault. The results cited indicate that DALG will require impractical amounts of time to attain 100 percent test coverage for ECAT structures. While the results demonstrate that PODEM performs very effectively for ECAT structures, there is no proof that such performance is guaranteed for all ECAT structures. The test coverage on non-ECAT type circuits was generally the same for PODEM and DALG. However, the execution speed of PODEM was strikingly faster than that of DALG. Table II shows the test coverages and a normalized comparison of the execution speeds of PODEM and DALG as derived from the evaluation study. To obtain the data of Table II, both PODEM and DALG were first executed to generate tests for the stuck faults in each case. The tests were minimized by an automatic procedure that compacted the tests using a static compaction procedure [11]. The compacted tests were then fault simulated to obtain the test coverage. The test coverage of 31.2 percent obtained by DALG on case 3 should not be interpreted as indicating the percentage of faults for

which DALG actually generated tests. Actually, DALG generated tests for 14 faults only and the remainder of the tested faults were tested "accidentally."

Summary and Conclusions

An explanation has been developed for the poor performance of the D-algorithm in generating tests for stuck faults in the ECAT class of combinational logic circuits. The PODEM algorithm is described in detail and it has been demonstrated that PODEM solves the test generation problem faced by the D-algorithm. Study results are quoted to demonstrate that PODEM is significantly faster than the D-algorithm for the general class of combinational logic circuits. The similarities of the PODEM algorithm with other implicit enumeration algorithms, a subset of branch and bound algorithms, have been identified. The problem of test generation has been formulated as one involving the search of an n-dimensional 0–1 state space constrained by a set of Boolean equations and hence, amenable to an implicit enumeration approach. The description of PODEM uses a combinational logic schematic for test generation; however, it is evident that the technique used in PODEM is applicable for solving a set of generalized Boolean equations.

References

[1] S. G. Chappell, "Automatic test generation for asynchronous digital circuits," *Bell Syst. Tech. J.*, vol. 53, pp. 1477–1503, Oct. 1974.
[2] J. P. Roth, "Diagnosis of automata failures: A calculus & a method," *IBM J. Res. Develop.*, vol. 10, pp. 278–291, July 1966.
[3] E. Balas, "An additive algorithm for solving linear programs with zero-one variables," *Oper. Res.*, vol. 13, pp. 517–546, 1965.
[4] F. F. Sellers, M. Y. Hsiao, and L. W. Bearnson, "Analyzing errors with the Boolean difference," *IEEE Trans. Comput.*, vol. C-17, pp. 676–683, July 1968.
[5] N. J. Nilsson, *Problem-Solving Methods in Artificial Intelligence*. New York: McGraw-Hill, 1971, ch. 3.
[6] E. W. Lawler and D. E. Wood, "Branch-and-bound methods—A survey," *Oper. Res.*, vol. 14, pp. 669–719, 1966.
[7] R. Garfinkel and G. L. Nemhauser, *Integer Programming*. New York: Wiley, 1972, ch. 4, pp. 4–24.
[8] T. J. Snethen, "Simulator oriented fault test generator," in *Proc. 14th Des. Automat. Conf.*, June 1977, pp. 88–93.
[9] E. B. Eichelberger and T. W. Williams, "A logic design structure for LSI testability," in *Proc. 14th Des. Automat. Conf.*, June 1977, pp. 462–468.
[10] A. Yamada *et al.*, "Automatic system level test generation and fault location for large digital systems," in *Proc. 15th Des. Automat. Conf.*, June 1978, pp. 347–352.
[11] P. Goel and B. C. Rosales, "Test generation and dynamic compaction of tests," in *Proc. 1979 Annu. Test Conf.*, Cherry Hill, NJ.

Prabhakar Goel was born in Meerut, India, on January 16, 1949. He received the B. Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1970 and the M.S. and Ph.D. degrees in electrical engineering from Carnegie-Mellon University, Pittsburgh, PA, in 1971 and 1974, respectively.

Currently, he is a Development Engineer with the IBM Corporation and manages a group responsible for developing design automation tools for test generation and design verification. He has received two invention awards, an outstanding innovation award, and a corporate award for his work at IBM.