Journal of Circuits, Systems, and Computers, Vol. 3, No. 4 (1993) 859–875
© World Scientific Publishing Company

A SEARCH SPACE PRUNING METHOD FOR TEST PATTERN GENERATION USING SEARCH STATE DOMINANCE*

TAKAYUKI FUJINO
Department of Computer Science, Meiji University,
1-1-1 Higashimita, Tama-Ku, Kawasaki 214, Japan
HIDEO FUJIWARA
Graduate School of Information Science, Nara Institute of Science
and Technology, 8916-5 Takayama-Cho, Ikoma, Nara, 630-01 Japan

Received 8 June 1992
Revised 10 September 1993

In this paper, we present a new technique that can prune search space in test-pattern generation for combinational circuits. We extend the concept of search state equivalence derived by Giraldi and Bushnell, to that of search state dominance, and propose a new extended method, DST (Dominant STate hashing) algorithm, based on the search state dominance. The DST algorithm can prune the search space more effectively than the EST (Equivalent STate hashing) algorithm of Giraldi and Bushnell. Experimental results on benchmark circuits are reported.

1. Introduction

A significant amount of research has been devoted to finding more efficient algorithms for combinational logic test pattern generational.[1,3–12] Among those algorithms, the *D*-algorithm[3] was the first *complete* algorithm that could generate a test-pattern for any logical fault if such a test-pattern existed and enough computing time was given. The second significant progress in accelerating algorithms was achieved by PODEM[4] and FAN.[5,6] However, the computational resources required for test generation were still immense, i.e. there still remained some aborted faults in the ISCAS'85 benchmarks[2] due to the limited computing time. Later, some approaches and improvements[9–12] were proposed and succeeded in handling all faults in the ISCAS'85 benchmarks by either generating a test-pattern or a redundancy proof. Among those approaches, Giraldi and Bushnell[12] proposed a new test generation method, called the EST (Equivalent STate hashing) algorithm, in which search states were saved for all faults during test generation to prune the search space by using the information about previously visited states. The EST algorithm

---

*This paper was recommended by Associate Editor H. Yasuura.

has a characteristic feature such that it is orthogonal to all existing test generation algorithms, so it can be used to accelerate any test-pattern generator.

In this paper, we present a new technique that can prune the search space in test-pattern generation for combinational circuits. We extend the concept of *search state equivalence* derived by Giraldi and Bushnell, to that of *search state dominance*, and propose a new extended method, DST (Dominant STate hashing) algorithm, based on the search state dominance. The DST algorithm can prune the search space more effectively than the EST (Equivalent STate hashing) algorithm. First, we introduce the EST algorithm of Giraldi and Bushnell and come concepts; search state, evaluation frontier, and search state equivalence. Then, we extend the concept of search state equivalence to search state dominance, and then present some theorems and the DST algorithm. We illustrate the benefits of DST through examples of decision trees. Experimental results on benchmark circuits are also reported.

2. The EST Algorithm

Combinational logic circuits are tested by applying a sequence of input patterns that produce erroneous responses when faults are present and then comparing the responses with the correct (expected) ones. Such an input pattern or a primary input (PI) assignment used in testing is called a *test-pattern*. The problem of generating a test-pattern for a given fault can be viewed as a finite space search problem of finding a point in the search space that corresponds to a test-pattern. Test generation algorithms like PODEM[4] make a series of primary input (PI) assignments for fault sensitization and propagation in the circuit. When a fault is sensitized, a D-frontier appears in the circuit, where a *D-frontier* is defined as a set of gates with unspecified outputs and some input signal set to D or $\overline{D}$. Further PI assignments for advancing the D-frontier to a primary output (PO) of the circuit find a test-pattern. During this process, backtracking occurs when either the D-frontier disappears or when the fault site is not sensitized. In this way, test generation algorithms usually build a decision tree and apply a backtracking search procedure. Each node or step in the decision tree corresponds to a partial PI value assignment. Implication or five-valued (0, 1, X, D, $\overline{D}$) logic simulation for the partial PI assignment forms a decomposition in the circuit. Since each node in the decision tree corresponds to a search state, the search state is defined as the logic circuit decomposition derived from the PI assignment corresponding to the decision step. Figures 1(a), (b) and (c) show decompositions for sensitized (unsensitized) faults.

Giraldi and Bushnell[12] introduced *evaluation frontier (E-frontier)* to represent a search state more efficiently. The E-frontier represents a complete circuit cut-set labeling and uniquely identifies a circuit decomposition. In a five-valued (0, 1, X, D, $\overline{D}$) test generation algorithm, an E-frontier consists of all internal nets labeled with values other than X (unassigned) that are connected to the circuit POs by a path of gates with unassigned values (an X-path). In Fig. 1(a), the PI assignment is $\{x2 = 0\}$, and the E-frontier is $\{x5 = D, x7 = 1\}$. In Figs. 1(b) and (c), those PI assignments are $\{x2 = 0, x3 = 0\}$ and $\{x1 = 1, x2 = 1\}$, respectively, and have the same (equivalent) E-frontier $\{x6 = 1, x7 = 1\}$. Subsequent operations on such equivalent circuit decompositions produce equivalent results, and hence we define *search state equivalence* as follows: search states are called equivalent if their E-frontiers are equivalent.

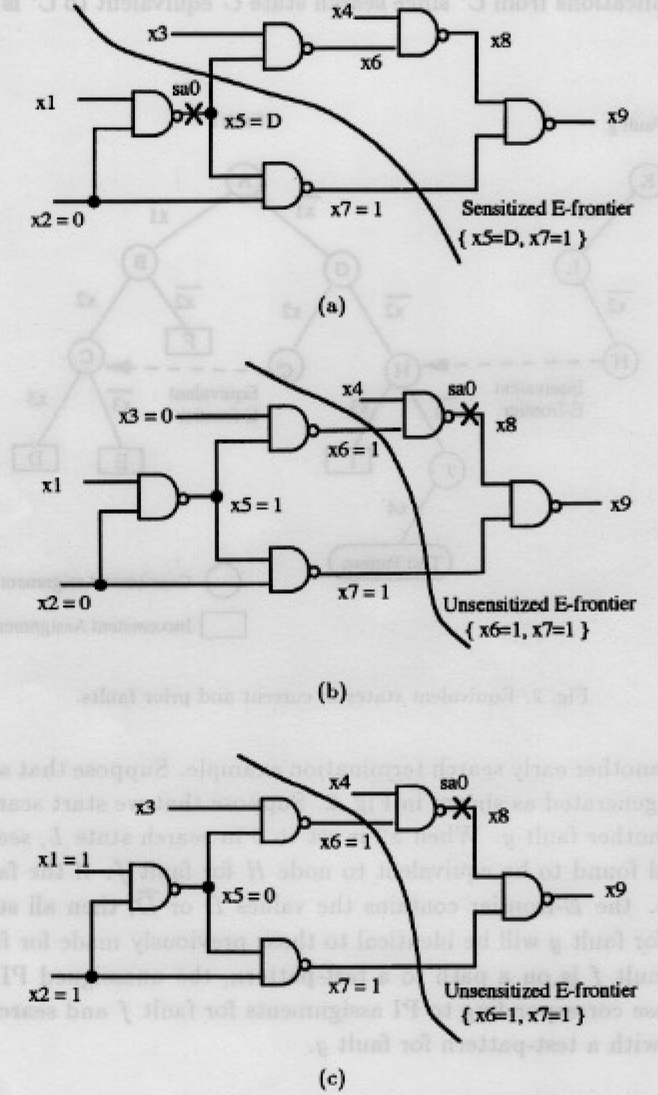


Fig. 1. Decompositions for Sensitized and Unsensitized faults (a) sensitized fault (b) unsensitized fault (c) unsensitized fault.

Giraldi and Bushnell proposed an approach to early identification of search path termination conditions by using E-frontiers, i.e. pruning search space. Consider the incomplete decision tree of Fig. 2 representing the search space for fault f. Node letters represent search states at each decision step. E-frontiers are computed at each node. Search starts at the root node A and proceeds in depth-first search order up to node G with implication resulting in search state C'. Suppose that the E-frontier of C' is equivalent to that of C. Then we can back up (backtrack) without exploring implications from C' since search state C equivalent to C' is known to be inconsistent.

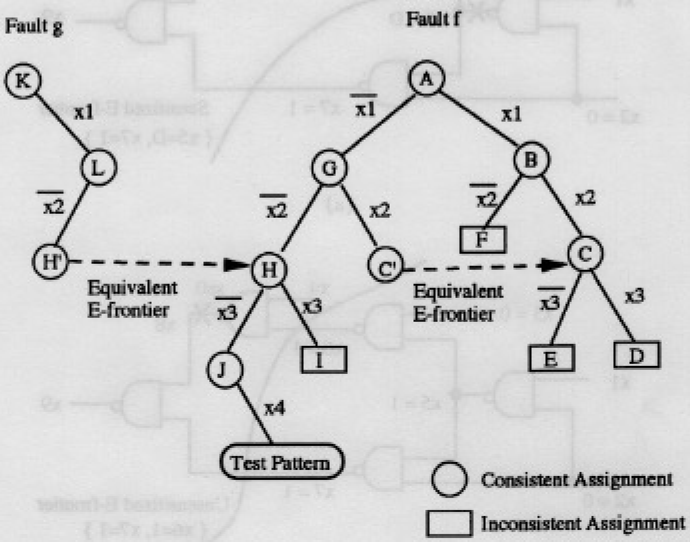


Fig. 2. Equivalent states in current and prior faults.

Consider another early search termination example. Suppose that a test-pattern for fault f is generated as shown in Fig. 2. Suppose that we start searching a test-pattern for another fault g. When $x2$ is set to 0 in search state L, search state H' is created and found to be equivalent to node H for fault f. If the fault has been sensitized, i.e. the E-frontier contains the values D or $\overline{D}$, then all subsequent PI assignments for fault g will be identical to those previously made for fault f. Since node H for fault f is on a path to a test-pattern, the unassigned PI's for fault g are set to those corresponding to PI assignments for fault f and search terminates immediately with a test-pattern for fault g.

3. Search State Dominance

In this section, we extend the concept of search state equivalence to that of search state dominance and present theorems for search path termination.

An E-frontier E can be represented by a set of pairs of net N and its value v, i.e. $E = \{(N_1, v_1), (N_2, v_2), \ldots, (N_k, v_k)\}$ or $E = \{N_1 = v_1, N_2 = v_2, \ldots, N_k = v_k\}$. Let E_i and E_j be E-frontiers. We say that E_i ***dominates*** E_j if

(1) any pair (N, v) in E_i is included in E_j (i.e. $E_i \subseteq E_j$), and

(2) any pair (N, v) in E_j such that $v = D$ or $\overline{D}$ is included in E_i (i.e. both E_i and E_j contain the same D-frontier).

For example, consider four E-frontiers, $E_1 = \{x1 = 0, x2 = 1\}$, $E_2 = \{x1 = 0, x2 = 1, x3 = 1, x4 = 0\}$, $E_3 = \{x1 = 0, x2 = 1, x3 = 1, x4 = \overline{D}\}$, and $E_4 = \{x1 = 0, x3 = 1, x4 = \overline{D}\}$. E_1 dominates E_2 since $E_1 \subseteq E_2$. However, E_1 does not dominate E_3 since D-frontiers of E_1 and E_3 are not the same. On the other hand, E_4 dominates E_3 since $E_4 \subseteq E_3$ and both E_4 and E_3 contain the same D-frontier $(x4 = \overline{D})$.

An E-frontier E is said to be ***sensitized*** if it contains value D or $\overline{D}$, i.e. the D-frontier of E is not empty. An E-frontier E is said to ***have a solution*** if there is a path from the node of the E frontier to a node of a test-pattern in the decision tree.

In the following, we present two theorems for early identification of search path termination. The first is the theorem in the case of searching a test-pattern for the same target fault.

Theorem 1: Let E_i and E_j be E-frontiers for the same target fault f. If E_i dominates E_j and E_i has no solution, then E_j has no solution.

Proof: Here we shall consider the PODEM algorithm[4] as the base test generation algorithm for the DST algorithm. We can similarly give proof of this theorem for other base test generation algorithms.

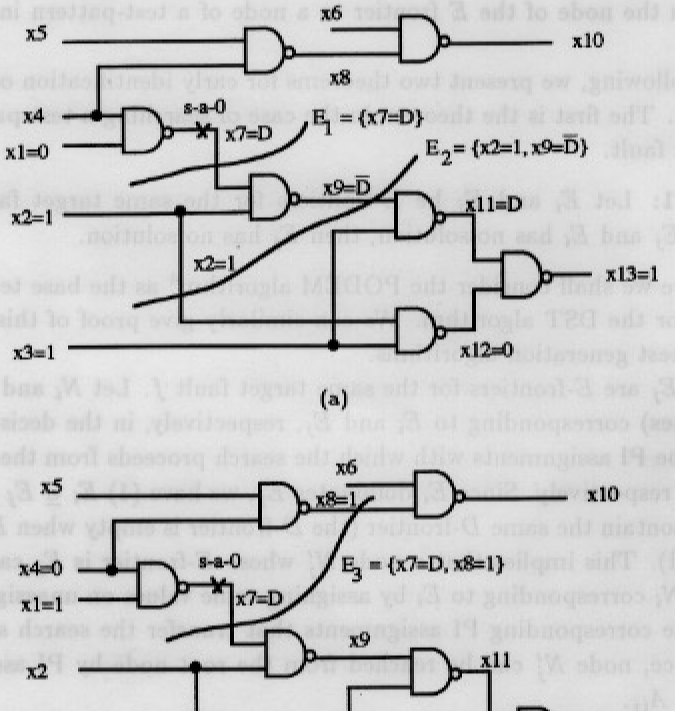
E_i and E_j are E-frontiers for the same target fault f. Let N_i and N_j be nodes (search states) corresponding to E_i and E_j, respectively, in the decision tree. Let A_i and A_j be PI assignments with which the search proceeds from the root node to N_i and N_j, respectively. Since E_i dominates E_j, we have (1) $E_i \subseteq E_j$ and (2) both E_i and E_j contain the same D-frontier (the D-frontier is empty when E_i and E_j are unsensitized). This implies that a node N'_j whose E-frontier is E_j can be reached from node N_i corresponding to E_i by assigning some values on unassigned PIs. Let A_{ij} be those corresponding PI assignments that transfer the search state from E_i to E_j. Hence, node N'_j can be reached from the root node by PI assignments A_i followed by A_{ij}.

Suppose that E_i has no solution for fault f but E_j has a solution for the same fault f. Since E_j has a solution for fault f, node N_j corresponding to E_j in the decision tree is on a path to a test-pattern for fault f. Let A_{jt} be those corresponding PI assignments that transfer from node N_j to the node where a test-pattern has been generated. The resulting test-pattern is the PI assignments A_j followed by A_{jt}. Since N'_j has the same E-frontier as N_j, node N'_j is also on a path to a test-pattern for fault f. Furthermore, since N'_j can be reached from N_i, node N_i is also on a path to a test-pattern for fault f. That is, the PI assignments A_i followed by

A_{ij} and A_{jt} can be the test-pattern for fault f. This contradicts that E_i has no solution for fault f. Hence, if E_i dominates E_j and E_i has no solution, then E_j has no solution.

Q.E.D.

Example 1: Let us try to generate a test-pattern for a fault $x7$ *s-a-0* in the circuit of Fig. 3(a). Let us first consider a conventional approach to searching a test-pattern for the fault. The decision tree is shown in Fig. 4(a). First we must set $x7 = 1$ to activate the fault $x7$ *s-a-0*. To justify $x7 = 1$ we first try $x1 = 0$. This implies $x7 = D$. This state corresponds to node 1 in Fig. 4(a) and the E-frontier is $E_1 = \{x7 = D\}$. To propagate the error or D-drive, we set $x2 = 1$, which implies $\{x2 = 1, x9 = \overline{D}\}$ (node 2 in Fig. 4(a)) and the E-frontier $E_2 = \{x2 = 1, x9 = \overline{D}\}$. To D-drive further, we try to set $x3 = 1$. However, this leads $x11 = D$, $x12 = 0$



(a)

(b)

Fig. 3. E-frontiers in test generation (a) search state of node 3 in Fig. 4 (b) search state of node 7 in Fig. 4.

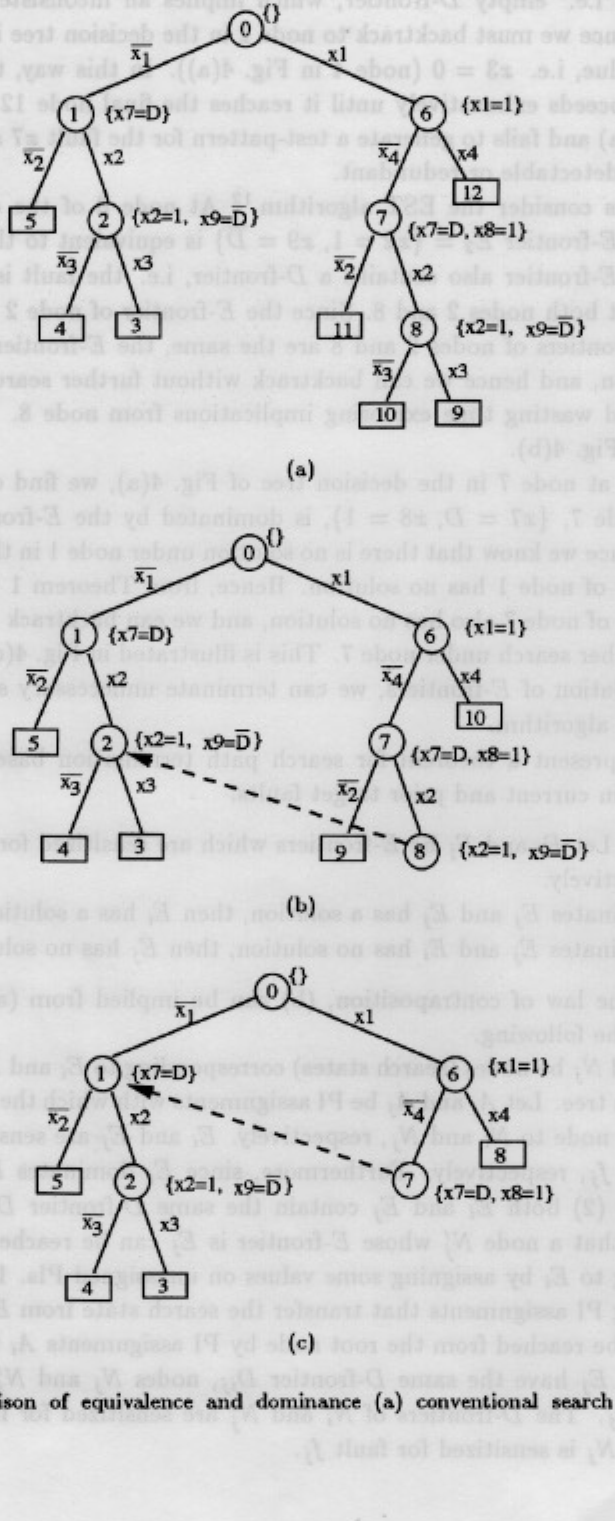


Fig. 4. Comparison of equivalence and dominance (a) conventional search (b) equivalence (c) dominance.

and $x13 = 1$, i.e. empty D-frontier, which implies an inconsistency (node 3 in Fig. 4(a)). Hence we must backtrack to node 2 in the decision tree in Fig. 4(a) and reverse the value, i.e. $x3 = 0$ (node 4 in Fig. 4(a)). In this way, the test-pattern generation proceeds exhaustively until it reaches the final node 12 of the decision tree in Fig. 4(a) and fails to generate a test-pattern for the fault $x7$ *s-a*-0. The fault $x7$ *s-a*-0 is undetectable or redundant.

Next let us consider the EST algorithm.[12] At node 8 of the decision tree in Fig. 4(a) the E-frontier $E_2 = \{x2 = 1, x9 = \overline{D}\}$ is equivalent to the E-frontier at node 2. The E-frontier also contains a D-frontier, i.e. the fault is sensitized and propagating at both nodes 2 and 8. Since the E-frontier of node 2 has no solution and both E-frontiers of nodes 2 and 8 are the same, the E-frontier of node 8 also has no solution, and hence we can backtrack without further search from node 8. EST can avoid wasting time exploring implications from node 8. This process is illustrated in Fig. 4(b).

If we look at node 7 in the decision tree of Fig. 4(a), we find out that the E-frontier of node 7, $\{x7 = D, x8 = 1\}$, is dominated by the E-frontier of node 1, $\{x7 = D\}$. Since we know that there is no solution under node 1 in the decision tree, the E-frontier of node 1 has no solution. Hence, from Theorem 1 we can see that the E-frontier of node 7 also has no solution, and we can backtrack from nodes 7 to 6 without further search under node 7. This is illustrated in Fig. 4(c). By using the dominance relation of E-frontiers, we can terminate unnecessary searching earlier than the EST algorithm.

Next, we present a theorem for search path termination based on dominant search states in current and prior target faults.

Theorem 2: Let E_i and E_j be E-frontiers which are sensitized for target faults f_i and f_j, respectively.
(a) If E_i dominates E_j and E_j has a solution, then E_i has a solution.
(b) If E_i dominates E_j and E_i has no solution, then E_j has no solution.

Proof: By the law of contraposition, (b) can be implied from (a). So, we shall prove (a) in the following.

Let N_i and N_j be nodes (search states) corresponding to E_i and E_j, respectively, in the decision tree. Let A_i and A_j be PI assignments with which the search proceeds from the root node to N_i and N_j, respectively. E_i and E_j are sensitized for target faults f_i and f_j, respectively. Furthermore, since E_i dominates E_j, we have (1) $E_i \subseteq E_j$ and (2) both E_i and E_j contain the same D-frontier D_{ij} (not empty). This implies that a node N_j' whose E-frontier is E_j can be reached from node N_i corresponding to E_i by assigning some values on unassigned PIs. Let A_{ij} be those corresponding PI assignments that transfer the search state from E_i to E_j. Hence, node N_j' can be reached from the root node by PI assignments A_i followed by A_{ij}. Since E_i and E_j have the same D-frontier D_{ij}, nodes N_j and N_j' have the same D-frontier D_{ij}. The D-frontiers of N_i and N_j' are sensitized for fault f_i, and the D-frontier of N_j is sensitized for fault f_j.

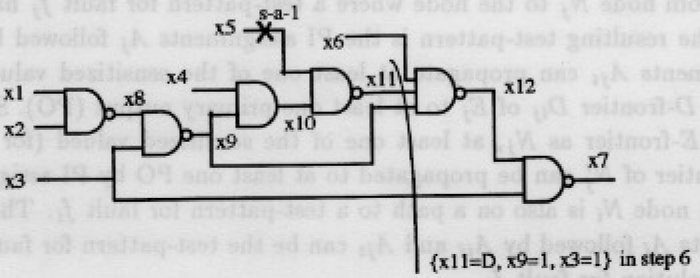
Since E_j has a solution for fault f_j, node N_j corresponding to E_j is on a path to a test-pattern for fault f_j. Let A_{jt} be those corresponding PI assignments that transfer from node N_j to the node where a test-pattern for fault f_j has been generated. The resulting test-pattern is the PI assignments A_j followed by A_{jt}. The PI assignments A_{jt} can propagate at least one of the sensitized values (for fault f_j) in the D-frontier D_{ij} of E_j to at least one primary output (PO). Since N'_j has the same E-frontier as N_j, at least one of the sensitized valued (for fault f_i) in the D-frontier of N'_j can be propagated to at least one PO by PI assignments A_{jt}. Therefore, node N_i is also on a path to a test-pattern for fault f_i. That is, the PI assignments A_i followed by A_{ij} and A_{jt} can be the test-pattern for fault f_i. Hence E_i has a solution for fault f_i.

Q.E.D.

Example 2: Let us consider two faults, $x5$ *s-a*-1 and $x10$ *s-a*-1, in the circuit of Fig. 5. First we consider to generate a test-pattern for the fault $x5$ *s-a*-1. In Fig. 5(a), the test-pattern generation process for fault $x5$ *s-a*-1 is illustrated. The test-pattern is $(x1, x2, x3, x4, x5, x6) = (1, 1, 1, 1, 0, 1)$.

Consider the subsequent search for fault $x10$ *s-a*-1 in Fig. 5(b). The test-pattern generation proceeds as indicated in Fig. 5(b). The corresponding decision tree is shown in Fig. 6(a). According to the EST algorithm, only the equivalence relation among E-frontiers is checked. Hence, in this example, at Step 6 in Fig. 5(b) the computed E-frontier, $\{x11 = D, x9 = 1, x3 = 1\}$, is found to be identical to the E-frontier in Step 6 of the decision tree for the fault $x5$ *s-a*-1 (Figs. 5(a)). The E-frontier contains D-frontier. Hence, all subsequent PI assignments for fault $x10$ *s-a*-1 is identical to those previously made for fault $x5$ *s-a*-1. Since node 6 for fault $x5$ *s-a*-1 is on a path to a test-pattern (Fig. 6(a)), the unassigned PIs for fault $x10$ *s-a*-1 are set to those corresponding PI assignments for fault $x5$ *s-a*-1, i.e. $x6 = 1$, and search terminates immediately with a test-pattern for fault $x10$ *s-a*-1, $(x1, x2, x3, x4, x5, x6) = (1, 1, 1, 0, 1, 1)$.

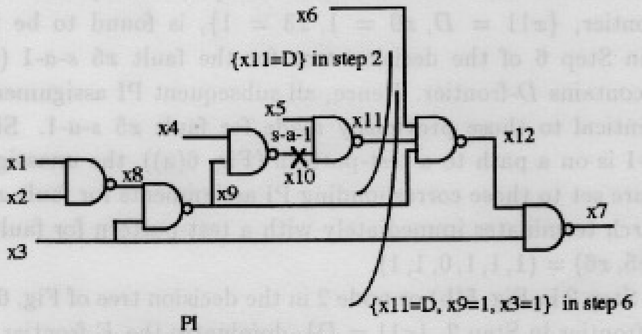
If we look at Step 2 in Fig. 5(b) or node 2 in the decision tree of Fig. 6(b), we find out that the E-frontier in Step 2, $\{x11 = D\}$, dominates the E-frontier, $\{x7 = D\}$, in Step 6 of the decision tree for the fault $x5$ *s-a*-1 (Fig. 5(a)). In Fig. 6(b), node 6 for fault $x5$ *s-a*-1 is on a path to a test-pattern. Therefore, from Theorem 2 we can see that fault $x10$ *s-a*-1 also has a test-pattern. The test-pattern is immediately obtained by setting the unassigned PIs for fault $x10$ *s-a*-1 to those corresponding PI assignments for fault $x5$ *s-a*-1, i.e. $(x1, x2, x3, x6) = (1, 1, 1, 1)$. The PI assignment at node 2 in the decision tree for fault $x10$ *s-a*-1 is $(x4, x5) = (0, 1)$. Hence the test-pattern for $x10$ *s-a*-1 is $(x1, x2, x3, x4, x5, x6) = (1, 1, 1, 0, 1, 1)$. Figure 6 shows the comparison of two decision trees based on search state equivalence and dominance. We can see that if we use the dominance relation of E-frontiers, we can reduce the size of search space and hence the time to search the space more effectively than the EST algorithm.



{x11=D, x9=1, x3=1} in step 6

Step	PI Assignment	E-frontier	
1	x5=0	{ x5=$\overline{D}$ }	
2	x3=0	{x7=1}	← Backtrack
3	x3=1	{ x5=$\overline{D}$, x3=1}	
4	x1=1	{ x5=$\overline{D}$, x3=1, x1=1}	
5	x2=1	{ x5=D , x9=1, x3=1}	
6	x4=1	{x11=D, x9=1, x3=1}	
7	x6=1	{x7=D}	← Test-pattern generated

(a)



{x11=D} in step 2

{x11=D, x9=1, x3=1} in step 6

Step	PI Assignment	E-frontier	
1	x4=0	{x10=$\overline{D}$}	
2	x5=1	{x11=D}	← Dominates step 6 in (a)
3	x3=0	{x7=1}	← Backtrack
4	x3=1	{x11=D, x3=1,}	
5	x1=1	{x11=D x3=1, x1=1}	
6	x2=1	{x11=D, x9=1, x3=1}	← Equivalent to step 6 in (a)
7	x6=1	{x7=D}	← Test-pattern generated

(b)

Fig. 5. Test generation for (a) $x5$ s-a-1 and (b) $x10$ s-a-1 faults.

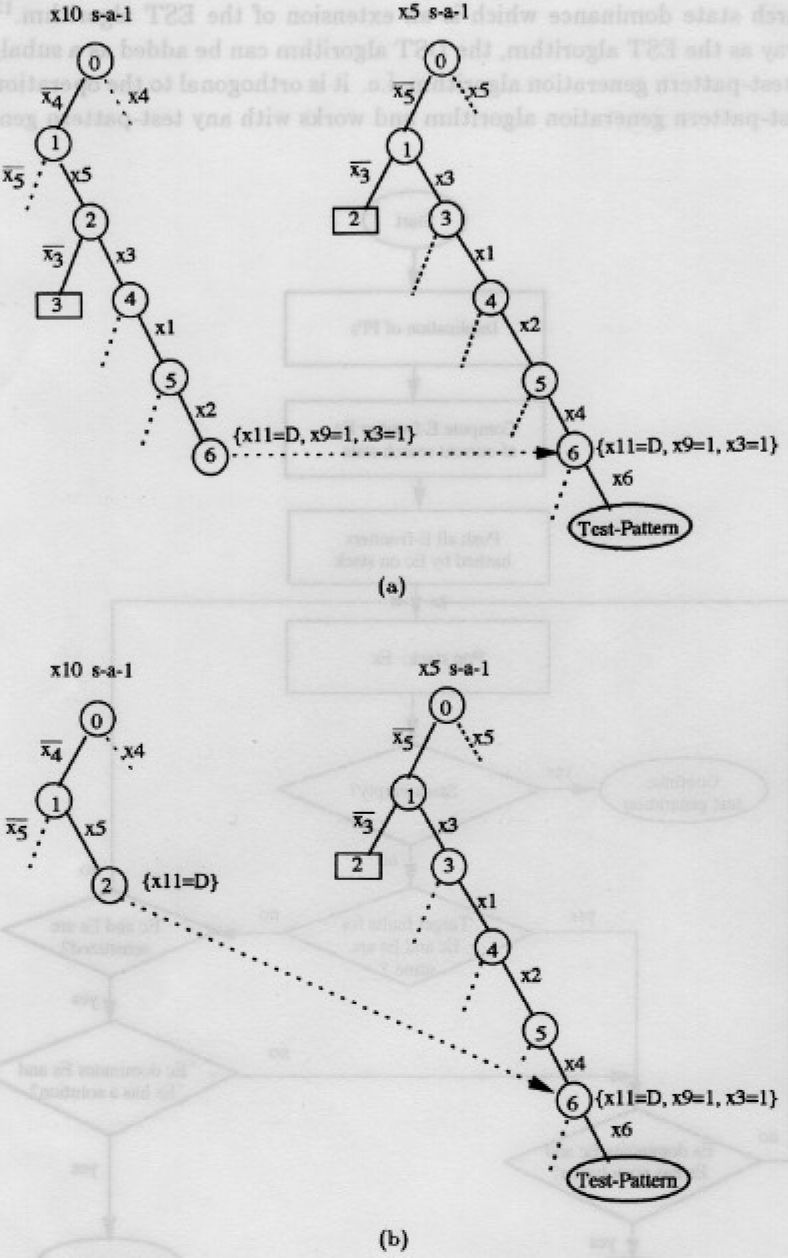


Fig. 6. Comparison of (a) equivalence and (b) dominance.

4. The DST Algorithm

In this section, we present the DST (Dominant STate hashing) algorithm based on the search state dominance which is an extension of the EST algorithm.[12] In the same way as the EST algorithm, the DST algorithm can be added as a subalgorithm to any test-pattern generation algorithm, i.e. it is orthogonal to the operations of the base test-pattern generation algorithm and works with any test-pattern generation.

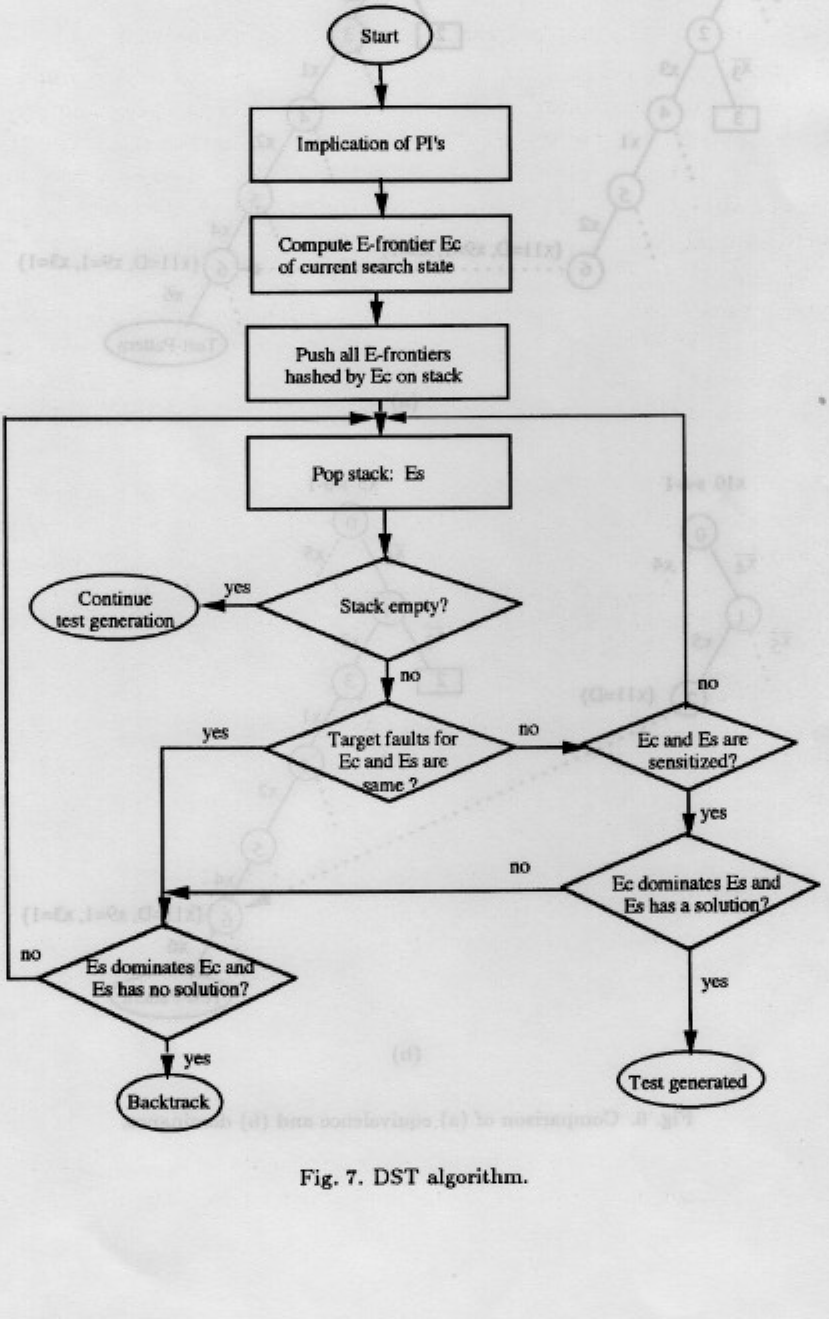


Fig. 7. DST algorithm.

Figure 7 shows the DST algorithm. A hash table is used to determine search state dominance. Each E-frontier is stored in the hash table with the data associated with the E-frontier, which includes the target fault, the solution flag (solution/no solution) and the test-pattern (if it exists). After implications in the base test generation algorithm, the DST algorithm starts and ends in one of the following three cases:

(1) exit with a test-pattern,
(2) exit to backtrack, or
(3) exit to continue the base test generation algorithm normally.

After starting the DST algorithm, each new E-frontier is computed and hashed into the hash table as follows: Let k_i be the total sum of index-numbers of all signal lines that have a faulty signal, D or $\overline{D}$, in an E-frontier E_i. For example, consider $E_1 = \{x1 = 0,\ x2 = \overline{D},\ x3 = D,\ x4 = 1\}$, then $k_1 = 2 + 3 = 5$. Here, we use this sum k_i as a *key* of the E-frontier E_i in hashing. If E_i dominates E_j, then both E_i and E_j contain the same D-frontier, i.e. they have the same faulty signals (D or $\overline{D}$), and hence $k_i = k_j$, i.e. they have the same key. For example, consider two E-frontiers, $E_1 = \{x1 = 0,\ x2 = \overline{D},\ x3 = D,\ x4 = 1\}$ and $E_2 = \{x1 = 0,\ x2 = \overline{D},\ x3 = D,\ x4 = 1,\ x5 = 0\}$. E_1 dominates E_2 and they have the same keys, i.e. $k_1 = k_2 = 5$. As a hash function h, we adopt the most commonly used method for hashing, i.e. to choose M to be prime and, for any key k, compute $h(k) = k \bmod M$.

All E-frontiers that dominate or are dominated by the current E-frontier Ec are taken from the hash table by hashing the key of Ec and then by checking whether each hashed E-frontier dominates or is dominated by Ec. Then they are pushed on a stack. Note that this stack is generated for each implication of PIs. If the stack is empty, the base test generation algorithm continues normally. While the stack is not empty, each stack entry is examined. Let Es be the E-frontier of the stack entry.

In case that Es is for the same fault as Ec, we further examine Es as follows: If Es dominates Ec and Es has no solution, the algorithm exits to backtrack (Theorem 1). Otherwise, the algorithm pops the stack and continues the stack loop. In the case that Es is for a different fault from Ec, we further examine as follows: If neither Ec nor Es is sensitized, the base test generation algorithm continues normally. If Ec and Es are both sensitized and if Ec dominates Es and Es has a solution, the test-pattern is formed and the unstacking loop is exited with a test-pattern for the current fault (Theorem 2(a)). If Ec and Es are both sensitized and if Es dominates Ec and Es has no solution, the algorithm exits to backtrack (Theorem 2(b)).

5. Experimental Results

The EST and DST algorithms have been implemented with PODEM[4] in the C programming language on a Sun-4/330, a 16 MIPS machine with a 32 Megabytes of memory. We have compared the resulting performance of PODEM and PODEM+DST, without any random-pattern generation or fault simulation. We have used no additional heuristics other than the Equivalent State Hashing and Dominant State Hashing algorithms. The backtrack limit is 1000. The results are given in Tables 2–4. The characteristic of the ISCAS'85 benchmark circuits used herein is shown in Table 1.

Table 1. Characteristics of ISCAS'85 benchmark circuits.

Circuit	#PIs	#POs	#Gates	#Faults	#Detectable faults	#Redundant faults
C432	36	7	160	524	520	4
C499	41	32	202	758	750	8
C880	60	26	383	942	942	0
C1355	41	32	546	1574	1566	8
C1908	33	25	880	1879	1870	9
C2670	233	140	1193	2747	2630	117
C3540	50	22	1669	3428	3291	137
C5315	178	123	2307	5350	5291	59
C6288	32	32	2406	7744	7710	34
C7552	207	108	3512	7550	7419	131

Table 2. Frequency of equivalent/dominant state hashing.

Circuit	#Hash tests			#Hash backtracks		
	Equivalence	Dominance	Diff	Equivalence	Dominance	Diff
C432	75	113	38	1981	1984	3
C499	212	236	24	0	0	0
C880	206	247	41	0	0	0
C1355	672	728	56	0	0	0
C1908	770	800	30	25	25	0
C2670	986	1145	140	3200	9328	6128
C3540	648	818	120	3476	4543	1067
C5315	859	1291	432	65	100	35
C6288	1059	1527	468	0	0	0
C7552	1067	1240	173	220	506	286

Table 2 compares the frequency of *E*-frontier matching in equivalent state hashing and dominant state hashing for PODEM+DST. #Hash Tests is the number of test-patterns found by equivalent/dominant state hashing. #Hash Backtracks is the number of backtracks found by equivalent/dominant state hashing. Diff is the difference between the dominance and equivalence values; the value of the dominance column minus the value of the equivalence column. Hence the column Diff shows the effect of the dominant state hashing over the equivalent state hashing, i.e. the number of early search terminations that cannot be found by the equivalent state hashing. From Table 2, the DST algorithm has a higher possibility of early search termination than the EST algorithm, especially the circuits C2670, C3540 and C7552.

Table 3 shows the comparison of the number of backtracks, the number of implications and CPU time for all faults targeted. #Backtracks-PODEM (PODEM+DST) shows the total number of backtracks for all faults in PODEM (PODEM+DST). #Backtracks-Difference shows the total number of backtracks in PODEM minus that in PODEM+DST. #Backtracks-Difference hence shows the effect of backtrack reduction caused by search space pruning. Similarly, #Implications-PODEM (PODEM+DST) shows the total number of implications for all faults in PODEM (PODEM+DST). #Implications-Difference shows the total number of implications in PODEM minus that in PODEM+DST. #Implications-Difference hence shows the effect of implication reduction caused by search space pruning.

Table 3. Performance comparison of search space pruning.

Circuit	#Backtracks			#Implications			CPU time (sec)	
	PODEM	PODEM +DST	Difference	PODEM	PODEM +DST	Difference	PODEM	PODEM +DST
C432	44020	44020	0	50313	49362	951	353.8	1085.8
C499	8688	8660	28	35958	27846	8112	727.7	1012.0
C880	1	1	0	8525	7214	1311	114.4	107.4
C1355	8482	8342	140	64676	39304	25372	1309.8	1293.1
C1908	9118	9078	40	35305	27723	7582	878.9	1221.1
C2670	152801	84378	68423	193798	110000	83798	7173.3	12480.5
C3540	210508	208769	1739	249083	242395	6688	11522.9	30681.6
C5315	16422	15730	692	82349	69214	13135	6563.0	6733.6
C6288	278941	278941	0	471419	450955	20464	43580.9	219649.8
C7552	160329	154661	5668	323505	296648	26857	35121.9	241023.4

In spite of the success in pruning search space, PODEM+DST requires however more CPU time than PODEM. This is due to the time consuming process of information saving and retrieval with memory constraints. We set the maximum size of the hash table, i.e. the maximum number of *E*-frontiers to be 50000. For large circuits, this limit was insufficient and more storage was required. Since the main purpose of this experiment is to show the superiority of DST on pruning search space, we have implemented a very simple system with a primitive technique for hashing or information retrieval that is nothing but a prototype. Hence to resolve this problem, much faster and more efficient information retrieval techniques for search states should be adopted in the system.

Table 4 shows the performance comparison of fault coverage for PODEM and PODEM+DST with a backtrack limit of 1000. The results indicate that the fault coverage of PODEM+DST is higher than PODEM for circuits C2670, C3540 and C7552. This result also shows the effectiveness of the search space pruning in PODEM+DST.

Table 4. Performance comparison of fault coverage.

Circuit	#Detected faults		#Redundant faults		#Aborted faults		Fault coverage %	
	PODEM	PODEM +DST	PODEM	PODEM +DST	PODEM	PODEM +DST	PODEM	PODEM +DST
C432	482	482	0	0	42	42	91.98	91.98
C499	750	750	0	0	8	8	98.94	98.94
C880	942	942	0	0	0	0	100.00	100.00
C1355	1566	1566	0	0	8	8	99.49	99.49
C1908	1864	1864	6	6	9	9	99.52	99.52
C2670	2565	2619 *	58	80 *	124	48 *	95.49	98.25 *
C3540	3185	3188 *	74	74	169	166 *	95.07	95.16 *
C5315	5288	5288	55	55	7	7	99.87	99.87
C6288	7504	7504	32	32	208	208	97.31	97.31
C7552	7346	7346	59	62 *	145	142 *	98.08	98.12 *

6. Conclusions

We have presented a new test-pattern generation algorithm based on a new concept called search state dominance. Search state dominance is an extended concept of search state equivalence introduced by Giraldi and Bushnell.[12] We have presented some techniques which can prune search space during test-pattern generation. Some theorems have been shown to guarantee the effectiveness of the search space pruning. The DST algorithm has been described and can be added to any test-pattern generation algorithm as a subalgorithm. We have finally presented the experimental results on ISCAS'85 benchmark circuits which show that the DST algorithm has the high possibility of pruning search space more effectively than the EST algorithm of Giraldi and Bushnell.[12]

References

1. H. Fujiwara, *Logic Testing and Design for Testability*, MIT Press, 1985.

2. F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits", *Proc. IEEE Int. Symp. Circuits and Systems: Special Session on ATPG and Fault Simulation*, June 1985.
3. J. P. Roth, "Diagnosis of automata failures: A calculus and a method", *IBM J. Res. Develop.* **10** (1966) 278–291.
4. P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits", *IEEE Trans. Comput.* **C-30** (1981) 215–222.
5. H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms", *IEEE Trans. Comput.* **C-32** (1983) 1137–1144.
6. H. Fujiwara, "FAN: A fanout-oriented test pattern generation algorithm", *Proc. IEEE Int. Symp. Circuits and Systems*, June 1985, pp. 671–674.
7. T. Kirkland and M. R. Mercer, "A topological search algorithm for ATPG", *Proc. 24th ACM/IEEE Design Automation Conf.*, June 1987, pp. 502–508.
8. M. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test-pattern generation system", *IEEE Trans. Computer-Aided Des.* **7** (1988) 126–137.
9. M. Schulz and E. Auth, "Improved deterministic test-pattern generation with applications to redundancy identification", *IEEE Trans. Computer-Aided Des.* **8** (1989) 811–816.
10. T. Larrabee, "Efficient generation of test patterns using Boolean difference", *Proc. IEEE Int. Test Conf.*, Aug. 1989, pp. 795–801.
11. J. Rajski and H. Cox, "A method to calculate necessary assignments in algorithmic test pattern generation", *Proc. IEEE Int. Test Conf.*, Sept. 1990, pp. 25–34.
12. J. Giraldi and M. L. Bushnell, "EST: The new frontier in automatic test-pattern generation", *Proc. 27th ACM/IEEE Design Automation Conf.*, June 1990, pp. 667–672.