Estimating Stuck Fault Coverage in Sequential Logic Using State Traversal and Entropy Analysis

Soumitra Bose*
Design Technology, Intel Corp.
Folsom, CA 95630
bose@ieee.org

Vishwani D. Agrawal
Dept. of ECE, Auburn University
Auburn, AL 36849
vagrawal@eng.auburn.edu

Abstract

Stuck fault coverage estimation for sequential circuits relies on a time expansion model, where combinational techniques are employed for each time-frame. Faults that are hard to detect and require a particular sequence of states are often incorrectly estimated to be detected. This problem is more evident for designs that exhibit low coverage either due to low testability or insufficient vectors that fail to exercise the required sequence of states. This paper illustrates how a simple state traversal analysis can mitigate this problem. For circuits with large number of sequential elements, we propose an entropy based technique that collapses the state graph to be analyzed. Experimental results for larger ISCAS benchmarks show that this technique reduces the coverage estimation error by as much as 50%.

1 Introduction

Simulation [5, 15] of stuck faults is a resource intensive task and several acceleration techniques have been proposed [4, 7, 9, 8, 13]. Verification of industrial designs often involves simulation of large number of functional patterns that can be several million cycles long. It is estimated that a modern microprocessor design requires about a month to complete fault grading, a step that needs to be repeated when relatively minor changes are made during the design cycle.

Statistical fault coverage estimation [11] is an alternative to fault simulation, for which improvements in accuracy have been proposed [2, 6]. Using structural analysis of combinational logic, these improvements find faults that are guaranteed to remain undetected by a sequence [2]. Other than spatial correlation, temporal correlations arise due to biased (non-random) input sequences. Errors caused by such temporal correlations can be reduced by a modified data collection algorithm to remove biasing [6]. Both of these methods to remove spatial and temporal correlations analyze combinational logic, and improve the analysis within a combinational time-frame. They ignore the permissible sequence of states, which is the subject of this paper.

The iterative array time-frame expansion model remains the most common way to extend coverage estimation algorithms that were originally developed for combinational circuits, to sequential logic. Observabilities at latch outputs are assigned to respective latch inputs, before applying combinational algorithms to the previous time-frame. As demonstrated by examples in Section 2, errors arise due to invalid state sequences that are implied by such a method.

In this paper, valid state sequences are obtained by extracting a state graph during simulation. This graph is then used by the coverage estimation algorithm. Experimental results in Section 5 show that the estimation errors are significantly lower than those with the time-frame expansion model. At a minimum, combinational estimation algorithms need to be used for each state in this graph. This overhead makes the approach impractical once the size of the state graphs exceeds some threshold. The rest of this paper demonstrates how entropy-based approaches can reduce the size of the state graph. With reduced state graphs, fault coverage estimates still exhibit lower errors than the iterative array models, keeping a time complexity well below that of logic simulation. The improved estimate is still much less expensive than the exact fault simulation.

*S. Bose is now with UBS Investment Bank, 1285 Avenue of the Americas, New York, NY 10019.

1-4244-1128-9/07/$25.00

2 Previous Work and Motivation

For a stuck-at-1 (sa1) fault at signal node G, *statistical fault analysis* (Stafan) [11] evaluates a detection probability per vector, $d1(G)$, which is then used to evaluate a cumulative detection probability, $D1(G)$, over N vectors as

$$D1(G) = 1 - k[1 - d1(G)]^N \quad (1)$$

where k is a biasing factor. The above formula assumes that N vectors are statistically independent. The per vector detection probability, $d1(G) = c0(G) \times b0(G)$, is a product of two factors: (1) probability of fault excitation, $c0(G)$, and (2) conditional probability of fault effect propagation to an observation point, $b0(G)$, given that fault excitation has occurred. The latter is also referred to as zero-observability of gate G. For sa0 faults, an analogous formula involving quantities like $d0(G)$, $c1(G)$ and $b1(G)$ can be derived.

The probability of excitation of a sa1 fault at gate G is measured by the zero-controllability at G, estimated by counting the number of 0's and dividing by the total number of vectors N, i.e.,

$$c0(G) = zero\_count(G)/N \quad (2)$$

and, similarly,

$$c1(G) = one\_count(G)/N \quad (3)$$

The conditional fault effect propagation probabilities, $b0(G)$ and $b1(G)$, are evaluated through a backward traversal starting at primary outputs. These are initialized to 1 at every output that ever attains logic values 0 and 1, respectively. Otherwise, these probabilities are initialized to 0. For a three input OR gate, with inputs i_1, i_2 and i_3, and output o, the zero-observability at input i_1, $b0(i_1)$, is estimated as

$$b0(i_1) = b0(o) \times c0(o)/c0(i_1) \quad (4)$$

The factor $c0(o)/c0(i_1)$ is a measure of the conditional probability of $i_2 = i_3 = 0$, given that input i_1 has a logic value 0. The above formula assumes that the conditional sensitization of input i_1 to output o is independent of the sensitization of the gate output to some primary output of the circuit. Similarly, the one-observability of input i_1 is approximated as

$$\begin{aligned} b1(i_1) &= b1(o) \times Prob(i_2 = i_3 = 0 \mid i_1 = 1) \\ &= b1(o) \times \frac{Prob(i_1 = 1, i_2 = i_3 = 0)}{Prob(i_1 = 1)} \end{aligned}$$

Using the law of total probability, we obtain $S(i_1)$, the probability of sensitizing gate input i_1 to output o:

$$S(i_1) = Prob(i_1 = 1, i_2 = i_3 = 0) + c0(o)$$

where $c0(o) = Prob(i_1 = 0, i_2 = i_3 = 0)$. Next, substituting $Prob(i_1 = 1) = c1(i_1)$ and using the expression for $S(i_1)$, we get

$$b1(i_1) = b1(o) \times \frac{S(i_1) - c0(o)}{c1(i_1)} \quad (5)$$

Sensitization probability $S(i_1)$ is measured by counting the number of cycles for which inputs i_2 and i_3 together have non-controlling values, and dividing by the length of the input sequence. Similar formulas can be derived for other combinational gate types [11].

When vectors are biased, N in Equation 1 needs to be modified to N_{eff}, obtained by counting only those vectors where a new input combination is observed at any gate [6]. The observability evaluation in Equation 5 is also modified by changing the way $S(i_1)$ and other controllability counts are measured. For all *unique* state combinations seen at the input of the OR gate, counts are obtained from logic simulation. The counts for those input combinations that sensitize the input i_1 are added up and used as the sensitization count $S(i_1)$. The controllability measures $c0(o)$ and $c1(i1)$ are also obtained by adding the counts for the relevant input combinations only. Further details of this algorithm are skipped and the reader is encouraged to refer to the original paper [6].

Figure 1(a) shows a sequential circuit with a single latch that is assumed to have an initial logic state of 1. Three vectors are applied and logic values of all signals are shown in the diagram. The combinational time-frame used by existing algorithms is shown in the dotted line box. In this combinational circuit, the fanout stem driven by latch $G11$ has no dominator and reconvergence analysis cannot be utilized to improve accuracy [2, 6]. The primary output driven by $G9$ is always 0, and its 0-observability is 1. Tracing back to the inputs of $G9$, gate $G4$ has non-zero 1-observability because input combination 10 is observed at $G9$. Similarly, $G8$ has non-zero 1-observability. Tracing back from $G4$, the fanout stem driven by $G11$ has non-zero 0-observability. Similarly, tracing back from $G8$, $G7$ and $G5$, this fanout stem has non-zero 1-observability.

Figure 1(b) shows the time-frame expansion model

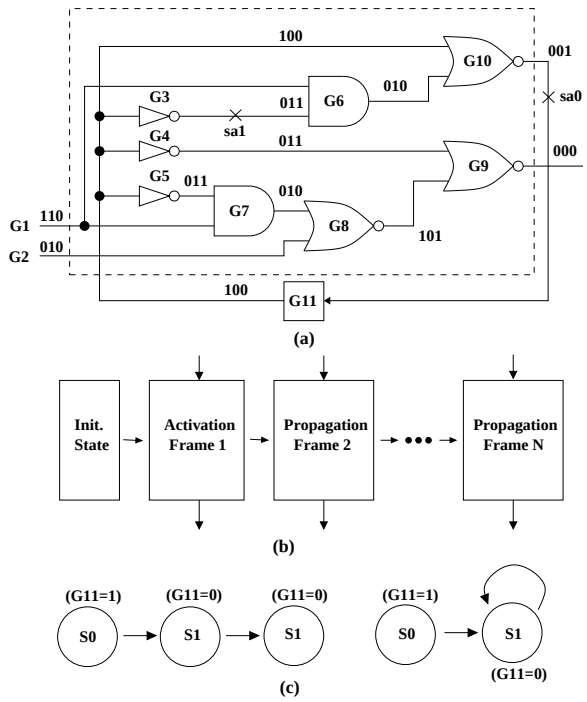


Figure 1: Illustration of over-estimation error.

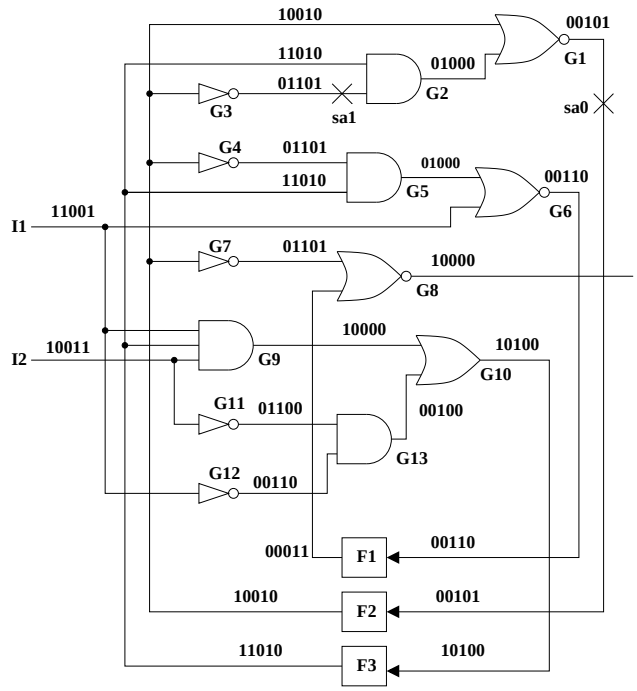


Figure 2: A five-vector over-estimation example.

used to estimate fault coverage for a sequential circuit. Fault activation is assumed to occur in the first time-frame. Once the 1-observability of the output of $G11$ is deduced to be non-zero, time-frame expansion implies the output of $G10$ to have non-zero 1-observability as well. The $sa0$ fault at the output of $G10$ is hence incorrectly estimated to be detected. The effect of this error propagates backwards beyond gates $G10$ and $G6$, and the $sa1$ fault at the output of $G3$ is also incorrectly estimated to be detected. This is due to the 00 and 10 input combinations seen at gates $G10$ and $G6$, respectively.

Figure 1(c) shows the states that were traversed during the three clock cycles. State $S0$ with $G11 = 1$ implies fault activation, necessarily in the previous cycle. Since $S0$ is the first state in the graph of Figure 1(c) and in never revisited again, the $sa0$ fault cannot be detected. Such state sequencing errors are common when simulation statistics are collected for individual time-frames, without regard to their ordering.

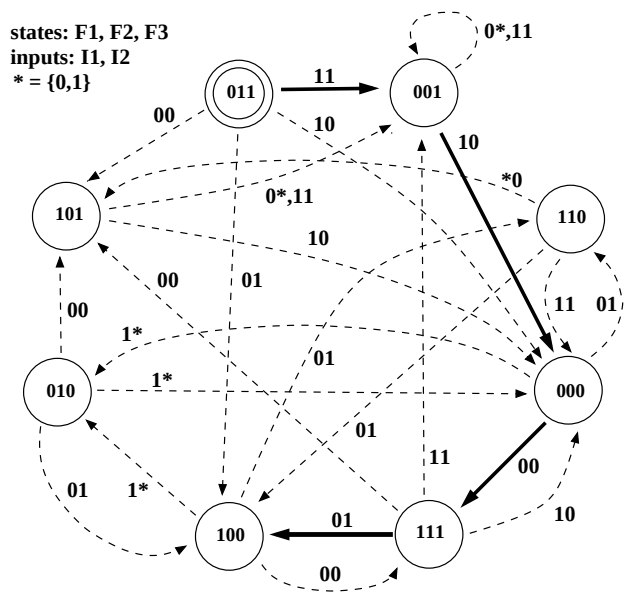
Figure 2 shows a more involved example of over-estimation error for a five-vector simulation. The reader may verify that the two faults shown in Figure 2 are incorrectly estimated as detected. The rest of the paper shows how the state sequencing information can help reduce estimation errors in sequential circuits. The example of Figure 2 will be used to illustrate the concepts presented in this paper.

3 State Traversal Analysis

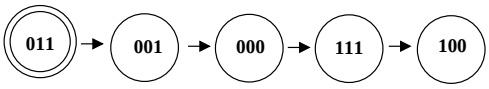
In a circuit with n sequential elements, each element is modeled by a variable, l_i, $0 \leq i \leq (n-1)$, that has one of three values: $\{0, 1, X\}$. The universe of all possible states is represented by $S = L_0 \times L_1 \times \ldots \times L_{n-1}$, where each $L_i = \{0, 1, X\}$. During simulation, only a subset of all possible states may be observed. The traversal of this subset of states is defined by a transition system, $M = \langle S, I, R \rangle$, where

- S is the set of states traversed during simulation
- I is the initial state of the circuit
- $R \subseteq S \times S$ is a transition relation

Note that M includes only the states that are traversed during simulation of a particular set of vectors. A transition system is not the same as a finite state machine [10], where edges (or input transitions) have input labels. For the example of Figure 2, the finite state machine for the circuit is shown in Figure 3(a), where 011 is the initial state. The labels on the arcs correspond to values of the



(a) Expanded FSM with traversed path.



(b) Transition system for above traversal.

Figure 3: State traversal for Figure 2

input signals, $I1$ and $I2$. The corresponding transition system M is shown in Figure 3(b). A transition system can be thought of as a non-deterministic state machine where the transition relation denotes the existence of a state transition for some input value.

A state transition system is easy to extract during simulation. Each observed state is matched with a set of previously seen states stored in a hash table. The hash table is a state transition system $M = \langle S, I, R \rangle$. If no match is found, the new state is added to the table, along with a transition relation to the new state. Transition relations are also added for previously seen states when no such relation exists in the hash table. The disadvantage of this technique lies in the number of states that may be observed while simulating a long sequence of vectors. Entropy measures [1, 14] can be used to reduce the size of the transition system extracted, as explained in the following paragraph.

For a signal s, entropy $H(s)$ is evaluated as

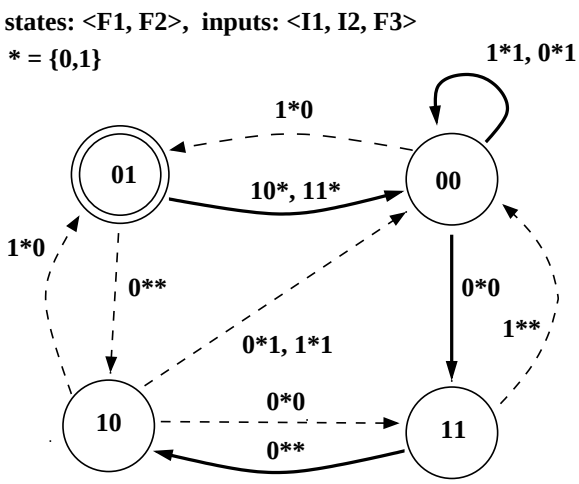
$$H(s) = -\sum_{i=0,1} p_i(s) log_2(p_i(s))$$

Step 1: E_{min} = threshold entropy for state collapsing
N_{sample} = entropy evaluation interval
Step 2: Vec# = 0
$S = \emptyset$, $R = \emptyset$, $M = \ < S, init\_state, R >$
Step 3: Previous state, PS = current circuit state
Simulate one cycle. Increment Vec#.
New state, NS = current circuit state
Step 4: If $NS \notin S$, $S = S \bigcup \{NS\}$
If $(PS, NS) \notin R$, $R = R \bigcup \{(PS, NS)\}$
Step 5: If (Vec# mod $N_{sample} = 0$), reduce M
Step 6: If more vectors, goto Step 3

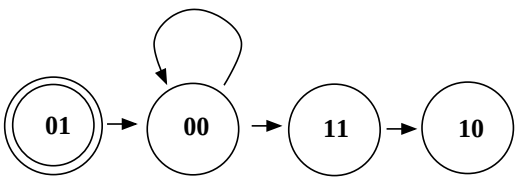
Figure 4: Algorithm for transition system extraction.

where $p_0(s)$ and $p_1(s)$ are probabilities for the signal to be, 0 and 1, respectively. The maximum value 1.0 for $H(s)$ is obtained for signals that are equiprobable, while the minimum value 0.0 is obtained for signals that are constant at 0 or 1. Entropy measures the degree of uncertainty of a signal and low values indicate signals that have high predictability. During fault coverage estimation, large errors are observed when such signals are implicitly assumed to have high observability for both logic values 0 and 1. While signal controllability can be obtained directly from a logic simulator, signal observability must be evaluated by an estimator and special care is needed to avoid large errors when signal entropy is low.

In addition to identifying signals that have low observability (and low controllability), entropy measurement can also help reduce the size of the transition system that is extracted by the simulator. Figure 4 shows the algorithm for state transition extraction, where entropy-based state collapsing is assumed to occur in Step 5 every N_{sample} cycles, using a threshold entropy of E_{min}. Any state variable whose entropy exceeds this threshold is effectively removed by merging states that differ in only that variable. This process is referred to as *reduce* M in the Step 5 of Figure 4. The merged states have a value X for the state element whose entropy exceeded the given threshold. Consider Figure 3(a), where all state elements have identical values of entropy. Assume that latch variable $F3$ is used for transformation, by collapsing all states that differ only in $F3$, and considering the eliminated state element as an additional input. Figure 5(a) shows the transformed finite state machine obtained from Figure 3(a). The sequence of bold edges in Figure 5(a) shows same traversal in this transformed machine as in Figure 3(a). The transition system corresponding to this minimization is shown in Figure 5(b).



(a) Collapsed FSM with traversed path.



(b) Transition system for traversal in (a).

Figure 5: Minimized state transition system.

4 Fault Coverage Estimation

We first introduce some notation before presenting the algorithm for fault coverage estimation. We assume the circuit has K latches, $\{l_1, \ldots, l_K\}$, where $N \leq K$ latches are used to describe each state of a transition system $M = \langle S, I, R \rangle$. Each state, $s_i \in S$, can be described by a set of logic values for these N latches: $\{l_1 = v_{i1}, \ldots, l_N = v_{iN}\}$. We use a vector notation to denote a vector of floating point numbers. We assume latch observabilities are represented by two vectors of floating point numbers, $\underline{z[i]}$ and $\underline{o[i]}$, $1 \leq i \leq K$, for 0 and 1 observabilities, respectively. We define the following restriction operators on these observability vectors, where $\Re$ is the set of real numbers:

- Zero-restricter: $(\underline{z \downarrow s_i}) : \Re^K \times S \rightarrow \Re^K$,

$$(\underline{z \downarrow s_i})[j] = \begin{cases} 0 & \text{if } v_{ij} = 1,\ j \leq N \\ z[j] & \text{if } v_{ij} = 0 \text{ or } j > N \end{cases}$$

- One-restricter: $(\underline{o \uparrow s_i}) : \Re^K \times S \rightarrow \Re^K$,

$$(\underline{o \uparrow s_i})[j] = \begin{cases} 0 & \text{if } v_{ij} = 0,\ j \leq N \\ o[j] & \text{if } v_{ij} = 1 \text{ or } j > N \end{cases}$$

Intuitively, the above restriction operators use a state vector and correct a set of possible zero (one) observabilities by setting those observabilities to zero that have their latches set to one (zero) in that state. In addition, we define a max operator:

- max: $\Re^K \times \Re^K \rightarrow \Re^K$

$$max(\underline{z_1}, \underline{z_2}) = \begin{cases} \underline{z_1}[j] & \text{if } \underline{z_1}[j] \geq \underline{z_2}[j] \\ \underline{z_2}[j] & \text{if } \underline{z_2}[j] > \underline{z_1}[j] \end{cases}$$

The algorithm for fault coverage estimation, shown in Figure 6, uses a global list of observabilities for each signal in the circuit. For ease of explanation, this global list of observabilities at latch outputs is referred to as $max0obs$ ($max1obs$) for logic value 0 (1). It traverses all states of the transition system using a procedure similar to depth-first search [3]. On reaching a new state, the algorithm determines if either observability increases for at least one state element. Unlike depth-first search, a state may be evaluated twice if the above condition is found to be true. If such a state element is found, the traversal is continued at all it's predecessor states. As in the iterative array model, a fault effect in any given combinational time-frame may either propagate directly to a primary output, or get latched in a state element for detection at a later cycle. The specific steps of the algorithm shown in Figure 6 are discussed next.

The state traversal procedure is initialized at the last state visited during simulation (line 1), and an implication procedure is then performed (line 2). The observabilities for detection at a primary output, for the last state, are stored in variables zPO and oPO (line 3). These observabilities are also used to initialize the global observabilities stored in variables $max0obs$ and $max1obs$ (line 4). The last state, along with the latch observabilities, are then added to a stack (line 5). States are removed from the stack in last-in-first-out order (line 6), and all its predecessor states are analyzed (line 7). For each of these predecessor states, a logic implication step is performed (step 8), with N latches fixed to their respective values, to determine constant signal values. For $N = K$, this is equivalent to simulation with fixed state elements and primary inputs set to X. A combinational time-frame observability evaluation is then per-

```
Procedure Evaluate_Observabilities(Transition_System < S, I, R >){
    Ns = last state in sequence;                                  // start search from last state                          ...(1)
    imply_state(Ns);                                              // find constant values before next step                 ...(2)
    (zPO, oPO) = PO_backtrace(PO_observabilities for Ns); // initialize latch to PO observabilities for Ns              ...(3)
    max0obs = zPO; max1obs = oPO;                                 // initialize global observability table                 ...(4)
    push_stack(Ns, zPO↓Ns, oPO↑Ns);                               // detection latency is 1 cycle for last state           ...(5)
    while (((<Ns, zNs, oNs> = pop_stack()) ≠ ∅) {                 // remaining states to be traversed                      ...(6)
       for each (Ps = previous state of Ns) {                     // analyze all predecessor states                        ...(7)
           imply_state(Ps);                                       // find constant values before next step                 ...(8)
           <z1,o1> = latch_backtrace(zNs, oNs);                   // find observabilities given previous state Ps          ...(9)
           (zPO, oPO) = PO_backtrace(PO_observabilities for Ps);     // find latch to PO observabilities for Ps          ...(10)
           <zPs, oPs> = <max(z1,zPO)↓Ps, max(o1,oPO)↑Ps>;     // add observabilities from 9 and 10                       ...(11)
           if ((<zPs,oPs> ≰ <maxOobs,max1obs>) OR                 // this observability exceeds global values              ...(12)
                                  (Ps not visisted before) {      // or predecessor state is new                           ...(13)
              push_stack(Ps, zPs, oPs);                           // add state to stack and update global                  ...(14)
              <max1obs,max0obs> = <max(max1obs,oPs), max(max0obs,zPs)>;     // observabilities                              ...(15)
           }
       }
    }
}
```

Figure 6: Observability evaluation using state transition system.

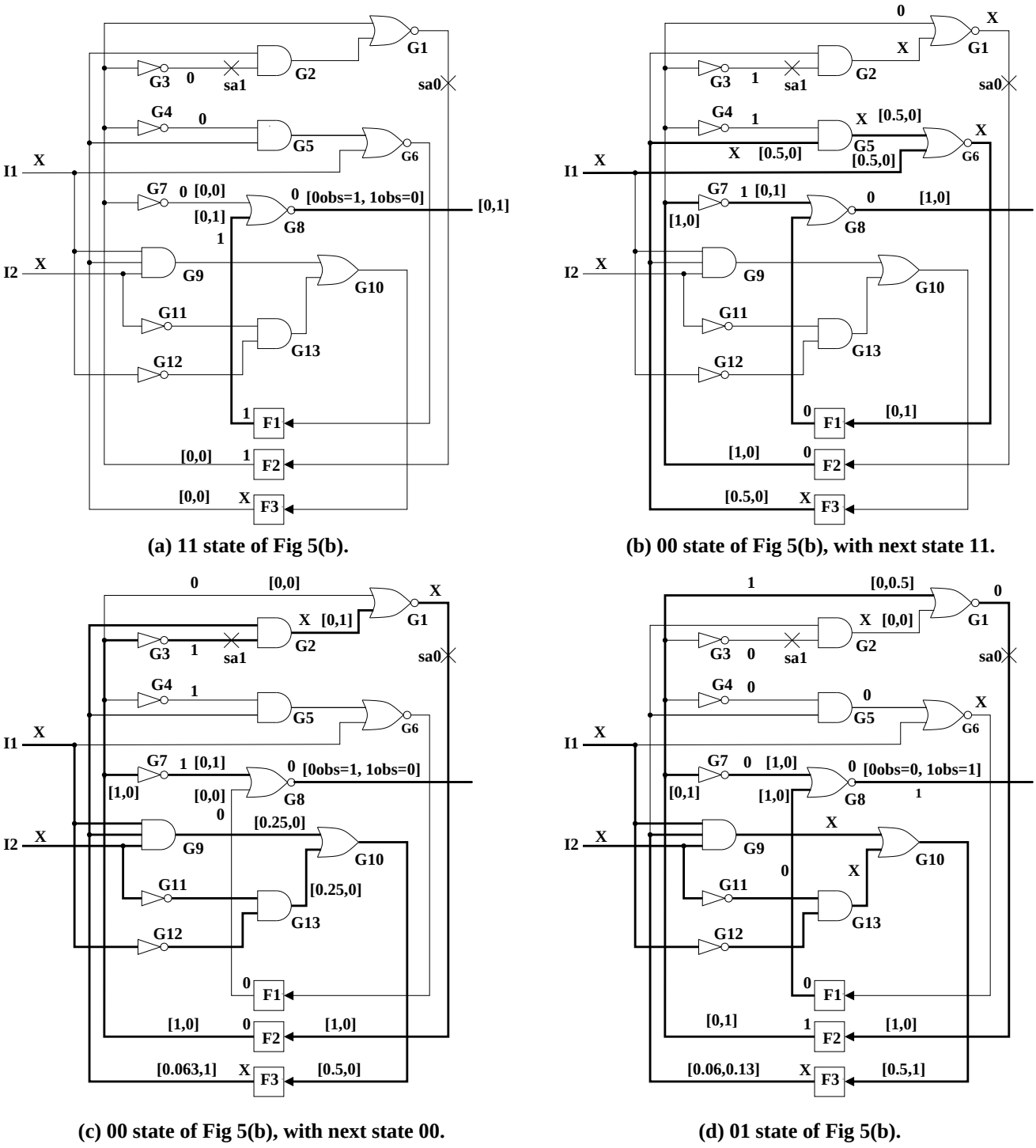
formed (steps 8 and 9), with the latch data *input* observabilities set to values equal to those obtained for latch outputs in step 6. The logic implication procedure in step 8 is expensive and may be almost equivalent to simulation if all state elements are stored in the state graph. The number of states in the transition system, and consequently the number of implication steps, also increase as N approaches K during the state extraction process of Figure 4. The new observabilities at a predecessor state are then added to the observabilities for detection at primary outputs (line 11), and then compared to the global observabilities (line 12). The predecessor state is added to the stack if this new state increases the global observabilities at any latch output (line 13), after which the global observabilities are also updated (line 14).

5 Results

We illustrate this algorithm for the circuit of Figure 2, using the transition system of Figure 5(b). Starting with the last state of the transition system of Figure 5(b) (state 10), an implication is performed (step 2 of Figure 6). Both inputs of $G8$ evaluate to logic 1, The backtrace procedure of step 3 stops at $G8$ since both inputs have controlling values. This state is then pushed onthe stack, with zero observability values for all latches (step 5). When this state is removed from the stack (step 6), its only predecessor state (11) is then analyzed (step 7). Analysis of this state is shown in Figure 7(a). The latch observabilities at latch outputs in successor states are moved to latch inputs, along with observabilities at primary outputs, and a backward observability evaluation is performed. Lines with non-zero observabilities for each logic value 0 or 1 are shown in bold in Figure 7.

Additionally, the observabilities for some of the lines, for logic 0 and 1, respectively, are shown as 2-tuples in square brackets in Figure 7. For the 11 state, all latch inputs have zero observabilities, while the primary output has an observability of 1 for logic 0 (since $G8$ evaluates to 0). The only latch with a non-zero observability is $L1$ for logic 1. The combinational observability evaluation for previous state 00 is shown in Figure 7(b). Two latches, $F2$ and $F3$, have non-zero observabilties in this state, as shown in Figure 7(b). Due to the self-loop at state 00 in the transition system of Figure 5(b), the state is analyzed again, as shown in Figure 7(c). The maximum values of the observabilities at latches $F2$ and $F3$ in Figures 7(b) and (c) are then used for state 01, shown in Figure 7(d). For detectability of the $sa0$ at $G1$, the maximum 1-observability observed at $G1$ is used. This maximal observability for logic 1 at $G1$ is zero, correctly implying the undetectability of this fault. The undetectability status of the $sa1$ fault at $G3$ is also correctly inferred by the algorithm.

Table 1 presents experimental data when the algo-



(a) 11 state of Fig 5(b).

(b) 00 state of Fig 5(b), with next state 11.

(c) 00 state of Fig 5(b), with next state 00.

(d) 01 state of Fig 5(b).

Figure 7: Tracing last three states in example of Figures 2 and 5(b).

Table 1: Iterative array and state space search comparison.

circuit (1)	exact cov (%) (2)	runtime (s) logic (3)	runtime (s) fault (4)	iterative array model est (%) (5)	iterative array model pos (%) (6)	iterative array model neg (%) (7)	iterative array model time (s) (8)	state exploration model est (%) (9)	state exploration model pos (%) (10)	state exploration model neg (%) (11)	state exploration model states (12)	state exploration model time (s) (13)
s298	83.7	0.14	0.87	84.04	0.35	0.00	0.13	83.43	0.35	0.96	382	0.29
s344	97.4	0.19	0.84	97.73	0.32	0.00	0.09	97.16	0.42	0.66	4323	0.69
s349	95.85	0.24	1.23	95.85	0.00	0.00	0.09	95.10	0.45	1.21	7621	0.93
s382	14.63	1.81	5.99	13.01	0.00	1.63	0.10	13.01	0.00	1.63	483	0.29
s386	66.11	0.58	0.93	68.06	1.94	0.00	0.07	66.28	1.11	0.94	30	0.10
s400	14.64	1.79	5.34	12.69	0.00	1.95	0.07	12.69	0.00	1.95	482	0.32
s420	72.92	0.56	6.86	76.96	4.99	0.95	0.12	76.96	4.99	0.95	9921	2.13
s444	13.77	1.99	6.34	11.74	0.00	2.03	0.07	11.74	0.00	2.03	504	0.35
s510	100.0	0.21	0.60	100.0	0.00	0.00	0.06	100.0	0.00	0.00	99	0.17
s641	86.61	0.39	2.62	87.04	0.65	0.22	0.09	86.83	0.43	0.22	1836	1.89
s1196	95.83	0.49	2.06	96.75	2.00	1.08	0.13	96.50	1.91	1.25	538	0.69
s1488	66.05	1.39	6.72	66.53	0.48	0.00	0.15	65.93	0.48	0.61	56	0.29
s5378	68.15	7.2	36.4	71.57	3.42	0.00	0.54	69.46	2.06	0.74	20000	42.53
s9234	33.57	16.32	78.78	41.52	8.37	0.43	0.70	36.13	3.32	0.76	20000	91.32
s13207	29.75	36.42	336.52	39.10	9.65	0.31	1.05	34.32	4.95	0.38	20000	266.4
s15850	41.10	66.12	264.39	46.17	4.24	0.04	1.13	44.13	2.79	0.61	20000	278.3
s35932	82.92	49.32	180.41	94.14	11.22	0.00	2.1	85.65	3.19	0.45	5286	192.6
s38417	11.94	98.06	2152.3	25.60	13.94	0.27	2.83	17.52	6.73	1.14	20000	362.51
s38584	62.84	127.2	668.4	71.54	8.93	0.29	2.57	66.60	5.49	1.79	20000	540.31
				average	3.71	0.48			2.03	0.96		
				std dev	4.51	0.69			2.17	0.57		

rithm outlined in this paper was used to estimate fault coverage for the ISCAS sequential benchmarks. All circuits were simulated with a random sequence of 20000 vectors, except s510 where fault coverage reached 100% after about 1300 vectors. Exact fault coverages for these sequences are shown in column 2.

Runtimes for logic and fault simulation are shown in columns 3 and 4, respectively. The logic simulation runtimes include the overhead to collect all statistics required for the estimation algorithm. All runtimes are reported in seconds for a Linux Pentium class machine. Columns 5-8 present data when an iterative array model is used for coverage estimation. Column 5 shows the fault coverage estimate, while columns 6 and 7 show the *positive* and *negative* errors, in percentages, respectively. The positive (negative) error is the percentage faults that are incorrectly classified as detected (undetected). These errors are meaningful measures of any statistical or probabilistic estimation method. Because a close to accurate coverage estimate can occur even when the two types of errors are large but balance each other.

Column 8 of Table 1 shows the runtime when a previously reported algorithm [2, 6], illustrated symbolically in Figure 1(b), was used. Note that this runtime includes the overhead to evaluate Equations 4 and 5 backwards through combinational logic, and then through sequential elements, as shown in Figure 1(b), till steady state probability values are obtained. This procedure is similar to finding the steady state probabilities in a discrete time Markov stochastic process [12]. Columns 9-13 show similar data for the state space exploration algorithm of this paper. No entropy based state collapsing was performed to obtain the data for columns 9-13. The last two rows show the average and standard deviation of the errors for the iterative array and state exploration models.

As evident from Table 1, the average and the standard deviation of the positive error decreases considerably with the state exploration algorithm. For the negative error, the average error increases slightly, though the variation in this error seems more predictable. Moreover, the average increase in the negative error, i.e., $0.96-0.48$, is much less than the average decrease in the positive error, i.e., $= 3.71 - 2.03$. However, these average measures are skewed by the smaller highly testable circuits where

Table 2: Accuracy-runtime trade off using entropy.

circuit (1)	entropy threshold=0.1 time (s) (2)	states (3)	cov (%) (4)	entropy threshold=0.2 time (s) (5)	states (6)	cov (%) (7)	entropy threshold=0.3 time (s) (8)	states (9)	cov (%) (10)	entropy threshold=0.4 time (s) (11)	states (12)	cov (%) (13)
s298	0.11	1	84.04	0.14	15	83.94	0.14	18	83.94	0.13	35	83.84
s344	1.49	1	97.83	1.49	19	97.43	1.54	131	97.23	1.54	143	97.23
s349	2.06	1	95.85	2.07	20	95.30	2.21	132	95.25	2.49	139	95.25
s382	0.15	5	13.01	0.16	5	13.01	0.15	8	13.01	0.15	18	13.01
s386	0.10	1	68.06	0.10	1	68.06	0.10	1	68.06	0.12	4	68.06
s400	0.17	5	12.69	0.15	5	12.69	0.16	8	12.69	0.15	18	12.69
s420	2.35	1	76.96	2.40	1	76.96	2.36	1	76.96	2.35	3	76.96
s444	0.15	2	11.74	0.15	8	11.74	0.15	5	11.74	0.15	12	11.74
s510	0.11	1	100	0.11	1	100	0.11	1	100	0.11	1	100
s641	0.41	1	87.04	0.39	1	87.04	0.45	10	86.99	0.47	89	86.86
s1196	0.14	1	96.75	0.14	1	96.75	0.22	42	96.75	0.38	165	96.58
s1488	0.17	1	66.53	0.17	1	66.53	0.17	1	66.53	0.17	1	66.53
s5378	3.86	278	71.25	4.21	305	71.32	5.01	471	71.39	6.52	556	71.39
s9234	13.82	47	39.44	13.62	55	38.65	20.01	263	37.55	35.60	4063	36.72
s13207	17.2	78	38.12	20.55	534	37.03	60.6	4429	35.57	135.2	12317	35.48
s15850	24.1	291	45.65	42.5	2507	44.85	58.2	3741	44.64	168.1	14850	44.23
s35932	8.4	19	94.10	10.25	23	92.99	12.1	147	92.98	25.65	1429	88.77
s38417	8.25	39	23.89	23.2	428	28.2	22.56	520	23.2	61.5	2175	15.82
s35854	23.2	515	71.17	54.9	1684	69.8	111.1	3553	69.01	222.9	7169	66.4

the iterative array model have acceptable errors. In general, estimates from the iterative array model have higher errors for circuits that either have low fault coverage or are not easily testable. If the last six circuits alone are considered, the positive error is found to decrease from 9.39% to 4.41% (not shown in the table, but verifiable from the last six rows). Compared to this 4.98% drop in the positive error, the negative error is found to increase marginally from 0.22% to 0.86%.

As evident from columns 8 and 13 of Table 1, the increased accuracy in the state exploration algorithm comes with a price. For the bigger circuits, the number of states traversed during simulation of the 20000 vectors approaches the length of the sequence. This happens when a new state is obtained for every vector. Column 12 of Table 1 shows the number of distinct states obtained when these circuits were simulated with these random sequences.

Table 2 presents coverage estimates obtained when the entropy based approach was used to reduce the number of states. Four cutoff entropy thresholds were used to collect this data: 0.1, 0.2, 0.3 and 0.4. State elements whose entropy exceed these thresholds are considered to have unknown (X) values when the state graph is constructed. Columns 3, 6, 9 and 12 show the number of states when such entropy-based state collapsing is performed. With fewer states, the coverage estimation time decreases, as shown in columns 2, 5, 8 and 11. With fewer number of states, the accuracy of the fault coverage estimates also decreases, as shown in columns 4, 7, 10 and 13. Though the overshoot and undershoot errors have not been shown in Table 2, their respective values lie in between the two values shown in columns 6-7 and 10-11 of Table 1. While entropy was used for thresholding in this experiment, the total number of states obtained after collapsing can also be used. Considering the original uncollapsed state graph, the state elements can be ordered by decreasing entropy. Those state elements at the top of the list are collapsed first. After collapsing with respect to a given element, the new size of the state graph can be obtained. Such a collapsing procedure can be continued till the size of the state graph falls below a threshold.

Higher values of entropy thresholds increase the size of the state graphs that need to be analyzed by the algorithm of Figure 6. This increase in the number of states manifests itself as an increase in runtime, as evident from columns 2, 5, 8, and 11 of Table 2. At the extreme, the

total number of states may equal the number of vectors simulated. Note that our formulation stores collapsed values of the sequential elements in the state graph, and ignores the values at the primary inputs of the circuit. For this method to yield coverage estimates with zero error, an algorithm that analyzes both latches and primary inputs is necessary. Such an algorithm would resemble a "fault simulator" that works by post-processing the results of a logic simulator, and would clearly be impractical both due to excessive runtimes and memory usage. Rather than inventing a new method of fault simulation, the goal of this research was to make improvements to the iterative array model used for coverage estimation. As explained in the previous paragraph, the maximum allowable size of the graph can be left to the user.

6 Conclusion

A typical design cycle consists of a final phase when fault simulation and scan selection are repeated till coverage goals are met. Each iteration of this design loop can be lengthy when designs have low coverage and the number of functional patterns is high. Fault coverage estimation can be an useful tool in such situations. However, when coverages are low, the inaccuracy of estimation methods is the biggest obstacle to overcome. When coverages are high due to a complete scan selection process, the scope for overestimation errors is considerably low. The work presented in this paper addresses the overestimation problem for designs that have low testability due to an incomplete scan selection process. This problem is best demonstrated by s38417 that has a coverage of 11.94%, while the iterative array estimate is 25.60%.

As demonstrated in this paper, the time-frame expansion model exhibits errors in coverage estimation due to incorrect state sequences that are implicitly assumed by such a method. A new method that analyzes a state graph extracted during simulation is proposed as an alternative. To reduce the overhead of backward observability propagation through each state of the graph at least once, an entropy based state reduction algorithm has been proposed. As shown by the results in Section 5, substantial accuracy improvement can be obtained with such reduced graphs. Approximately half the error in the coverage estimates can be eliminated with an additional overhead that is comparable to the time for logic simulation, and considerably less than that of accurate fault simulation.

References

[1] V. D. Agrawal, "An Information Theoretic Approach to Digital Testing," *IEEE Trans. Computers*, vol. C-30, pp. 582–587, Aug. 1981.

[2] V. D. Agrawal, S. Bose, and V. Gangaram, "Upper Bounding Fault Coverage by Structural Analysis and Signal Monitoring," in *Proc. 24th IEEE VLSI Test Symp.*, 2006, pp. 88–93.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Design and Analysis of Computer Algorithms*. Reading, Massachusetts: Addison-Wesley, 1974.

[4] K. J. Antriech and M. Schulz, "Accelerated Fault Simulation and Fault Grading in Combinational Circuits," *IEEE Trans. CAD*, vol. 6, no. 9, pp. 704–712, Sept. 1987.

[5] D. B. Armstrong, "A Deductive Method of Simulating Faults in Logic Circuits," *IEEE Trans. Computers*, vol. C-21, pp. 464–471, May 1972.

[6] S. Bose and V. D. Agrawal, "Fault Coverage Estimation for Non-Random Functional Input Sequences," in *Proc. International Test Conf.*, 2006. Paper 19.3.

[7] W. T. Cheng and M. L. Yu, "Differential Fault Simulation for Sequential Circuits ," *Jour. Electronic Testing: Theory and Applications*, vol. 1, no. 1, pp. 7–13, Feb. 1990.

[8] S. Gai and P. L. Montessoro, "CREATOR: New Advanced Concept in Concurrent Simulation," *IEEE Trans. CAD*, vol. 13, no. 6, pp. 786–795, June 1994.

[9] S. Gai, P. L. Montessoro, and F. Somenzi, "MOZART: A Concurrent Multilevel Simulator ," *IEEE Trans. CAD*, vol. 7, no. 9, pp. 1005–1016, Sept. 1988.

[10] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Reading, Massachusetts: Addison-Wesley, 1979.

[11] S. K. Jain and V. D. Agrawal, "Statistical Fault Analysis," *IEEE Design & Test of Computers*, vol. 2, no. 1, pp. 38–40, Feb. 1985.

[12] A. Leon-Garcia, *Probability and Random Process for Electrical Engineering*. Reading, Massachusetts: Addison-Wesley, 1994.

[13] T. M. Niermann, W. T. Cheng, and J. H. Patel, "PROOFS: A Fast Memory Efficient Sequential Circuit Fault Simulator," *IEEE Trans. CAD*, vol. 11, no. 2, pp. 198–207, Feb. 1992.

[14] K. Thearling and J. A. Abraham, "An Easily Computed Functional Testability Measure," in *Proc. International Test Conference*, Oct. 1989, pp. 381–390.

[15] E. G. Ulrich and T. Baker, "Concurrent Simulation of Nearly Identical Digital Networks," *Computer*, vol. 7, pp. 39–44, Apr. 1974.