FUNCTIONAL TEST GENERATION FOR FINITE STATE MACHINES

Kwang-Ting Cheng
Jing-Yang Jou

AT&T Bell Laboratories,
Murray Hill, NJ 07974

ABSTRACT – **In this paper, we describe a functional test generation method for finite state machines. A functional fault model, called single transition fault model, in the state transition level is used. In this model, a fault causes a single transition to a wrong destination state. A fault collapsing technique for this fault model is also described. For each state transition, a small subset of states is cleverly selected as the faulty destination states so that the number of modeled faults for test generation is minimized. Based on this fault model, we developed an automatic test generation algorithm and built a test generation system. The effectiveness of this method is shown by experimental results on a set of benchmark finite state machines. A 100% stuck-at fault coverage is achieved by our method for several machines and a very high coverage (>97%) is also obtained for other machines. In comparison with a gate level test generator STG3 [1], the test generation time is speeded up by a factor of 100.**

1. Introduction

Theoretical approaches to the test generation of finite state machine (FSM) have followed verification methods. The circuit is modeled by a state transition table. A checking experiment [2] is designed to distinguish such a machine from any other machines with the same or fewer number of states. This approach is shown to be impractical because of the long test sequence it produces.

It has been shown that the cyclic (feedback) structure is mainly responsible for the high complexity of gate-level sequential circuit test generation [3]. Most implementations of the finite state machines have very complex cyclic structures [4], even though they typically have a small number of flip-flops. Test generation of finite state machines at gate-level is thus a time-consuming process.

A finite state machine (FSM) is normally represented by its state transition graph. The vertices in the graph correspond to states of the FSM and the edges correspond to the state transitions. Each transition is associated with 4-tuple, <input label, source state, destination state, output label>. In this paper, we propose a test generation method at the state transition level. The method is based on a functional fault model, called *single transition fault model*, of FSM. In this model, a fault causes a single transition to a wrong destination state. Because the faulty transition may get into any state other than the good machine destination state, the number of modeled faults for each transition in the state table will be equal to the total number of states minus one. Thus, the size of fault list will be equal to $(N-1)M$ where N is the number of states and M is the number of transitions.

Any irredundant structural fault (e.g. single or multiple stuck-at faults, bridging faults, etc.) in the implementation of a sequential machine will cause some changes in its state transition graph. One or multiple transitions will be corrupted. A transition is corrupted if its output label or its destination state or both are changed. Among these three possible corruptions of a transition, namely, (1) wrong output label, (2) wrong destination state and (3) both, A type (2) corruption dominates the other two types of corruption. That is, a sequence that detects a fault which cause a type (2) corruption on a transition will also detect faults that cause type (1) and type (3) corruptions on the same transition. A test sequence for a fault causing a type (2) corruption on a transition consists of three subsequences: the initialization sequence that forces the machine to the source state of the faulty transition, the input label of the faulty transition that activates the fault ,and the propagation sequence that differentiates the good and the faulty destination states. The first two subsequences, i.e., the initialization sequence and the activation vector already produce different output labels if type (1) or type (3) corruption occurs. Thus, for test generation, only the transition faults that cause wrong desitination states are considered. In the following, we refer a single (multiple) transition fault to a fault that causes a single (multiple) transition(s) to wrong destination state(s) while still retains correct output label(s) on the faulty transition(s).

Since a defective sequential machine can be viewed as one in which a multiple transition fault of some arbitrary multiplicity is present, a test sequence that detects all multiple transition faults will detect all irredundant permanent physical defects. However, the analysis of multiple transition faults is too complex in almost all practical cases. A machine of M transitions and N states has $M(N-1)$ single transition faults but an extremely large number, $N^M - 1$, of possible multiple transition faults. The test sequence for all detectable single transition faults may not detect all detectable multiple transition faults due to the masking phenomenon. However, it can be shown analytically and empirically that, in general, the masking probability is very low (less than 1%). Thus, the quality of the test sequence for single transition faults is close to the sequence derived from checking experiments [5]. Therefore, we only consider the single transition faults.

The size of the single transition fault list, $(N-1)M$, may be still too large and unacceptable. We further reduce the number of single transition faults by fault collapsing. One of the key ideas in this paper is that, for each transition, instead of modeling $N-1$ faults, only a small subset of states is cleverly selected as the faulty destination states so that the number of modeled faults for test generation is minimized and the modeled faults dominate the remaining single transition faults.

A test generation system FTG for the single transition faults is implemented. According to our experimental results on a set of benchmark finite state machines, the length of the test sequence generated by FTG is comparable to that produced by a gate level sequential circuit test generator STG3 [1] for single stuck-at faults in a multi-level implementation of the machine synthesized by our FSM synthesis system. The test generation time, in average, is speeded up by a factor of 100, when compared to STG3. The test sequence generated by FTG covers 100% single stuck-at faults for several machines. A very high stuck-at fault coverage (>97%) is also obtained for all other machines. However, the sequences generated by STG3 for stuck-at faults only cover 10%-72% of the single transition faults.

In next section, we first introduce the *Shortest State Group Differentiating (SGD) Sequence Set*, based on which the faulty destination states are selected. The fault collapsing technique is described in Section 3. Section 4 describes the algorithm of computing the shortest SGD sequence set. The test generation and fault simulation procedures are described in Section 5. The implementation of the system and experimental results are presented in Section 6.

2. State Group Differentiating Sequence

Hennie's checking experiment is based on the *distinguishing sequence* (DS). The DS is defined as an input sequence that produces different output responses for each initial state of the FSM. The distinguishing sequence may not exist and the upper bound on the length, if it exists, is $(N-1)N^N$, where N is the number of states [6]. Sabnani and Dahbura [7] proposed a test generation procedure that is based on the *Unique Input/Output* (UIO) sequences, also called simple I/O sequences [8], of the FSM. The UIO sequence for a state s is defined as an input sequence that can distinguish between state s and the remaining states of the machine. In other words, we can determine whether the machine was in state s or not by observing the output responses of the input sequence. Some FSMs do not possess UIO sequences for every state. The UIO sequences are typically shorter than the distinguishing sequences even though no better upper bound is proven for UIO sequence by now.

In the procedure presented in this paper, we first compute the **State Group Differentiating (SGD) Sequence Set** for each state.

Definition 1: A SGD sequence, $T(s,SG)$, for a state s and a state group $SG=\{s_1,s_2,\ldots,s_i\}$ where $s \notin SG$, is an input sequence that can distinguish between state s and the states in SG.

Definition 2: The **SGD sequence set** for a state s is a collection of $K(s)$ SGD sequences, $T(s,SG_1)$, $T(s,SG_2)$,..., $T(s,SG_{K(s)})$, that s, SG_1, SG_2 , ... and $SG_{K(s)}$ are disjoint subsets of all states S and their set union is S.

Each sequence in the sequence set distinguishes between the given state s and a subset of S, SG_i. The collection of state s and the state groups SG_i,

$i=1,2,...,K(s)$, forms a partition on S. Notice that, in a reduced FSM, the SGD sequence set always exists for any state and may not be unique. A reduced FSM is a FSM that, for any two states S_i and S_j, there exists at least one input sequence that can distinguish between S_i and S_j. The shortest one among them is referred to the state-pair differentiating sequence of S_i and S_j and its length is upper-bounded by the number of states in the FSM. According to the above definition, the collection of the $N-1$ state-pair differentiating sequences, each of which differentiates between state s and one other state in S, is a SGD sequence set for state s. In this case, $K(s)$ is equal to $N-1$ and SG_i contains only one state for $i=1, 2,.., N-1$. The UIO sequence for a state s is also a SGD sequence set for s that contains only one sequence (i.e., K(s)=1) and SG_1 contains all states but s. Thus, the SGD sequence set may not be unique.

Definition 3: The **shortest SGD sequence set** for a state s is a SGD sequence set of state s that the sum of the lengths of the component SGD sequences in the set is minimum. That is, the value of $SUMT(s) = \sum_{j=1}^{j=K(s)} |T(s,SG_j)|$ is minimum, where $|T|$ denotes the length of sequence T.

Notice that the shortest SGD sequence set may not be unique either. We are interested in finding any one of them.

Example 1: Consider the state table of M_1 shown in Table 1.

Table 1 - State table of M_1

State	Input	
	0	1
A	B/0	C/1
B	B/0	D/0
C	D/1	C/1
D	A/1	B/0

This machine does not possess a distinguishing sequence. For state A, it does not have an Unique I/O sequence either. Since input 0 can distinguish between state A and states C and D, and input 1 can distinguish between state A and state B, SGD sequences T(A, {C,D})="0" and T(A, {B})="1" form a SGD sequence set for state A. The value of SUMT(A) is equal to 2. For this simple example, it can be easily identified that this sequence set is one of the shortest SGD sequence set for state A. Similarly, sequences T(B, {D})="0" and T(B, {A,C})="1" form a shortest SGD sequence set for state B; sequences T(C, {D})="1" and T(C, {A,B})="0" form a shortest SGD sequence set for state C; sequences T(D, {C})="1" and T(D, {A,B})="0" form a shortest SGD sequence set for state D.

It will become clear later that the length of the test vectors generated by our method is proportional to the value of $SUMT(s)$. The test length is thus minimized by finding the shortest SGD sequence set.

Theorem 1: For a reduced FSM, the upper bound of the minimum $SUMT(s)$ for any state s is $N(N-1)$.

Proof: Here we provide an existence proof. For a reduced FSM, there always exists a state-pair differentiating sequence between any two states. By definition, the collection of the $N-1$ state-pair differentiating sequences between state s and any other state in S is a SGD sequence set of state s. Since the upper bound of the state-pair differentiating sequence between any two states is N, the value of the SUMT(s) of this set is upper-bounded by $N(N-1)$. Q.E.D.

This bound is much smaller than the upper bound of distinguishing sequence and UIO sequence, which is $(N-1)N^N$ [7]. It can be shown that the problem of finding the shortest SGD sequence set can be reduced to a set covering problem and is NP-complete. In Section 4, we will describe an efficient heuristic algorithm of finding the shortest SGD sequence set.

3. Fault Collapsing

The test sequence for a single transition fault F_i consists of three subsequences: initialization sequence, activation sequence and fault propagation sequence. The initialization sequence brings the machine from current state to the source state of the faulty transition. The activation sequence contains only one vector that is the input label associated with the faulty transition. The fault propagation sequence is the state-pair differentiating sequence that differentiates the good destination state s from faulty destination states s' and, thus, propagate the fault effects to the output.

Definition 4: For a given state s, a state group SG, and their SGD sequence $T(s,SG)$, the **dominating state** Dom(s,SG) is a state $s' \in SG$ that the state-pair differentiating sequence between state s and s' is the same as $T(s,SG)$. The dominating state may not be unique.

Notice that, according to definition 1 and 4, the state-pair differentiating sequence of state s and a dominating state of SG can also distinguish between state s and the other states in SG. Thus, the sequence that detects the fault causing the transition to the dominating state of SG also detect faults causing the same transition to other states in SG. Therefore, for any transition with destination state s, only one fault that causes the transition to a dominating state for the entire state group SG is modeled. Suppose the shortest SGD sequence set of state s contains $K(s)$ sequences and each of them distinguishes between state s and a state group SG_i, $i=1,2,..,K(s)$. We model only $K(s)$ single transition faults, $F_1, F_2, \cdots F_{K(s)}$, for each transition with state s as the destination state. The faulty destination state of fault F_i is a dominating state $Dom(s,SG_i)$ selected from state group SG_i. These $K(s)$ faults dominate the other $N-1-K(s)$ single transition faults associated with this transition. Thus, the test sequences generated for the collapsed modeled faults will detect all single transition faults.

If we denote $fanin(s)$ as the number of state transitions whose destination state is state s and S as the set of all states, the total number of collapsed single transition faults is equal to $\sum_{s \in S} fanin(s)K(s)$.

Example 2: For machine M_1 shown in Table 1, we model two faults for each state transition with destination state A. One will cause this transition to state C (or state D, as both are dominating states.) and the other into state B. Similarly, there will be two faults for each transition with destination state B, C or D. Since $fanin(A)=1$, $fanin(B)=3$, $fanin(C)=2$ and $fanin(D)=2$, the total number of modeled faults is equal to $1 \cdot 2 + 3 \cdot 2 + 2 \cdot 2 + 2 \cdot 2 = 16$.

4. Computing Shortest SGD Sequence Set

Let's restate the problem of finding the shortest SGD sequence set as follows:

Problem: Given a FSM M of n states, for each state s, partition the remaining states into $K(s)$ non-overlapping groups, $SG_j(s)$, $j \in \{1,2,...K(s)\}$, and find a corresponding dominating state, $Dom(s,SG_j)$, such that:

(1) for each state group $SG_j(s)$, there exists an input sequence $T(s,SG_j(s))$ that can distinguish between s and states in $SG_j(s)$, and

(2) $SUMT(s) = \sum_{j=1}^{j=K(s)} | T(s,SG_j(s)) |$ is minimal.

It can be shown that the problem is NP-complete. In this section, we describe an efficient algorithm that can produce a near-optimum solution.

We solve this problem in two steps. In the first step, we compute all the shortest state-pair differentiating sequences between the given state s and every other state s_i by adopting breadth-first search on the modified successor tree.

Example 3: Consider machine M_2 with state table shown in Table 2. Table 3 shows all the Shortest State-Pair Differentiating Sequences (SSPDS) between state A and other states.

Table 2 - State table of M_2

State	Input	
	I_1	I_2
A	C/1	E/1
B	A/0	D/1
C	E/0	D/1
D	F/1	A/1
E	B/1	F/0
F	B/1	C/1

Table 3 - SSPDS between state A and other states.

Seq. No.	B	C	D	E	F
T_1	I_1	I_1	I_1I_1	I_2	I_2I_1
T_2			I_2I_2		I_2I_2

In the second step, we partition the states and find a dominating state of each state group based on these sequences. The algorithm of finding the dominating states for a given state s and their corresponding state-pair differentiating sequences is given in Fig. 1. The outputs of this algorithm are the $K(s)$ dominating states for state s, which is $\bigcup_{i=1}^{K(s)} Dom(s,SG_i)$, and their corresponding state-pair differentiating sequences, which is $\bigcup_{i=1}^{K(s)} T(s, SG_i)$. Once these outputs are obtained, $K(s)$ single transition faults are modeled for each transition with destination state s. The faulty destination states are those $K(s)$ dominating states. The corresponding state-pair differentiating sequences will be used as fault propagating sequences during test generation. The value of $SUMT(s)$ is thus equal to the sum of the lengths of the fault propagation sequences for faults associated with this transition.

In the description of the algorithm, the number of the shortest state-pair differentiating sequences between state s_i and s_j is denoted as $nT(s_i,s_j)$ and each of them is denoted as $T_k(s_i,s_j)$, $k = \{1,2,...,nT(s_i,s_j)\}$. Their lengths are denoted as $|T(s_i,s_j)|$. For example, for machine M_2, there are two shortest state-pair differentiating sequences between state A and D, $T_1(A,D)$="I_1I_1" and $T_2(A,D)$="I_2I_2" respectively. The value of $nT(A,D)$ is 2, so is the value of $|T(A,D)|$.

Definition 5: A vector I_i is covered by another vector I'_i, denoted as $I_i \sqsubseteq I'_i$, if both vectors have the same bit-width and for every bit in I_i with a value 0(1), the corresponding bit in I'_i has a value 0(1) or - (don't cares).

Definition 6: An input sequence T = "$I_1,I_2,..,I_n$" dominates another input sequence T' = "$I'_1,I'_2,..,I'_m$" if $n \geq m$ and $I_i \sqsubseteq I'_i$, $i = \{1,2,...m\}$

Example 4: For a 2-input machine, vector '00' is covered by vector '-0' ('00' $\subseteq$ '-0'). Sequence T = "00,11,00" dominates sequence T' = "–0,1–".

Example 5: Consider machine M_2 in Table 2 and its shortest state-pair differentiating sequences (SSPDS) of state A in Table 3. We first choose a state which has the longest state-pair differentiating sequence. Since both $|T(A,D)|$ and $|T(A,F)|$ are equal to 2, we randomly start with state D. Since $T_1(A,D)$="I_1I_1" dominates $T_1(A,B)$="I_1" and $T_1(A,C)$="I_1", and $T_1(A,F)$="I_2I_1" domoinates $T_1(A,E)$="I_2", the first partition found by the algorithm is P={{B,C,D}, {E,F}} and the corresponding $SUMT(A)$ is equal to 2+2=4. On the other hand, $T_2(A,D)$ dominates $T_1(A,E)$ and $T_2(A,F)$, and $T_1(A,B)$ dominates $T_1(A,C)$, an alternative partition is {{D,E,F}, {B,C}} and the corresponding $SUMT(A)$ is equal to 1+2=3. The algorithm shown above will find both partitions and report the latter one because of the smaller value of $SUMT(A)$.

Table 4 shows the results of our proposed partitioning algorithm on a set of MCNC benchmark finite state machines. The second and third columns show the number of states and the number of transitions of the state machines. If we don't perform fault collapsing, the number of faults, equal to $M(N-1)$, is shown in the fourth column. The fifth column shows the

```
Input:  State $s$,  US = S - {$s$},
          $nT(s,s_i)$,  where $i = 1,2,..,n$, and $s_i \neq s$,
          $T_j(s,s_i)$,  where $j = 1,2,.., nT(s,s_i)$.

Output:  (MinSUMT, Dominating states,
           State-pair Diff. Seq.) $\leftarrow$ Partition(US).

Procedure Partition(US)
  begin
    randomly choose a state $s_i \in US$, which has
       max $|T(s,s_i)|$;
    LUS = US - {$s_i$};
    For each $T_j(s,s_i)$, where $j = 1,2,..,nT(s,s_i)$
      begin
        $T[j] = \{T_j(s,s_i)\}$;
        $F[j] = \{s_i\}$;
        $SUMT[j] = |T_j(s,s_i)|$;
        $cover = Dominate(LUS,T_j(s,s_i))$;
        $LUS = LUS - cover$;
        if $(LUS \neq \emptyset)$
          begin
            $(RSUMT,RF,RT) \leftarrow Partition(LUS)$;
            $SUMT[j] = SUMT[j] + RSUMT$;
            $F[j] = F[j] \cup RF$;
            $T[j] = T[j] \cup RT$;
          end
      end
    $SUMT[v] = \min_{j=1}^{nT(s,s_i)} SUMT[j]$;
    return(SUMT[v],F[v],T[v]);
  end

Procedure Dominate(LUS,T)
  begin
    $cover = \emptyset$;
    For each $s_k \in LUS$
      begin
        if any $T_p(s,s_k)$ is dominated by T
          $cover = cover \cup \{s_k\}$;
      end
    return($cover$);
  end
```

Fig. 1: Algorithm of partitioning states.

number of modeled faults generated by our method. The number of collapsed stuck-at faults of a multilevel implementation is shown in the last column for comparison. It is worth noting that the number of modeled faults can affect the length of test sequence significantly. By comparing the fifth and the last columns of Table 4, we can expect that the test sequence generated by our method is comparable to that generated by a good gate-level sequential circuit test generator.

In a state table, the transitions having identical source state, destination state and output label can be collapsed into a single transition in the specification. For example, transitions <00, S1, S2, O1> and <01, S1, S2, O1> can be collapsed into <0-, S1, S2, O1>. According to our experience, it is a reasonable heuristic to consider the collapsed transition as a single transition for fault modeling to reduce the total test length while still obtain a high-fault-coverage test sequence. The number of transitions M given in Table 4 is the number of collapsed transitions.

Table 4 - Number of Modeled Faults

Name	N†	M‡	$M(N-1)$	No. of Modeled Faults	No. of Collapsed s.a. Faults
dk14	7	56	336	94	249
dk15	4	32	96	84	176
dk16	27	108	2808	292	541
dk17	8	32	224	70	167
dk512	15	30	420	48	168
ex1	18	121	2057	526	556
ex4	14	21	273	88	204
ex7	9	36	288	152	148
planet	48	115	5405	654	1098
opus	9	29	232	72	200
styr	30	166	4814	1039	1103
bbara	7	42	252	135	129
cse	16	91	1365	507	446
bou	31	59	1770	59	634
sand	32	184	5704	1227	1018

† number of states
‡ number of transitions

5. Test Generation and Fault Simulation

The test generation procedure for a single transition fault consists of three steps: (1) Initialization: bringing the machine from current state to the source state of the faulty transition, (2) Excitation: executing the faulty transition, and (3) State differentiation: applying the corresponding SGD sequence to differentiate the good state from the faulty states. In preprocessing, the transfer sequences between any state pair are computed. Thus, the initialization sequence is readily available during test generation. So is the state differentiating sequences.

The initialization sequences and the state differentiating sequences for different faults can be effectively overlapped by dynamically ordering the fault list during test generation. For example, if the source state of the faulty transition of the next target fault matches the current state of the good machine, the initialization sequence will not be needed for that fault.

The test length and test generation time can be further reduced by incorporating a functional fault simulator in the test generator. The fault simulator will record the current faulty state for each faulty machine and remove the faults detected accidentally. The information of the current faulty state for each fault is used to determine the next target fault. The fault whose current faulty state is different from the current good machine state is the preferred next target fault. In this case, the fault is already activated and, thus, initialization and excitation steps are not needed. If no such fault exists, the one whose source state of the corresponding faulty transition is the same as the current state of the good machine is selected as the next target fault. In this case, initialization step is not needed. Otherwise, the one whose source state of the corresponding faulty transition is closest to the good machine's current state is selected to minimize the length of initialization sequence.

6. Experimental Results

A functional test generation system, FTG, is implemented in the C language for use in the UNIX environment. The input to the system is the state table of the finite state machine in KISS format. The system consists of three major programs: a fault list generator, a test generator and a fault simulator. Table 5 shows some preliminary results on a set of MCNC benchmark finite state machines. The second and third columns show the test generation time and test length to cover all single transition faults. The CPU seconds are for SUN4/260. We then synthesize the machine in a multi-level form by an automatic FSM synthesis system. The fourth column in Table 5 shows the coverage of single stuck-at faults of the generated sequence on this implementation. For comparison, we ran STG3 on the implemented circuits. The run time, length of the test vectors and the fault coverage are also shown in Table 5. In the experiments, the redundant stuck-at faults are removed from the fault lists. A 100% stuck-at fault coverage is achieved by FTG for several machines and a very high coverage (>97%) is also obtained for other machines. The test length is slightly longer than that generated by STG3 but much shorter than that derived from checking experiment.

Table 5 - Results on benchmark machines

Circuit	FTG			STG3		
	CPU Sec.	Test len.	Fault cov.	CPU Sec.	Test len.	fault cov.
dk14	1	228	100.00	12	90	100.00
dk15	0.2	146	100.00	4	44	100.00
dk16	10	406	100.00	7176	322	97.58
dk17	0.4	86	100.00	9	66	100.00
dk512	0.5	89	100.00	92	85	100.00
ex1	33	422	98.38	309	205	96.38
ex4	0.3	63	99.50	57	93	100.00
ex7	2	158	99.31	314	58	99.18
planet	35	600	98.26	13334	544	97.15
opus	0.2	96	98.50	25	80	100.00
styr	204	964	97.18	20613	515	94.28
bbara	2	241	100.00	36	133	100.00
cse	45	880	97.86	9657	344	97.76
bou	1	135	97.83	411	259	98.89
sand	202	809	97.74	9106	376	97.64
average	35.7	354	98.97	4077	214	98.59

We also conducted experiments on the fault coverage of the single transition faults by the test of the single stuck-at faults generated by STG3. The results are given in Table 6. The coverage of the single transition faults varies from 10% to 72%. The results shown in Table 5 and 6 imply that the single transition fault test set covers far more than single stuck-at faults. Unfortunately, there is no fault simulator that can handle other types of faultslike multiple stuck-at faults, transistor faults, bridging faults, etc. available to us, we are not able to show the fault coverage of those faults by the single transition fault test set.

7. Conclusion

We proposed a test generation method at the state transition level. Even though only a small subset of single transition faults is considered for test generation and fault simulation, the vector sequence generated is able to detect all single transition faults. The test length is optimized through the selection of the state group differentiating sequences. According to the experimental results, the test generation time is shown *two order* faster than that of a gate-level sequential circuit test generator. A high fault coverage for stuck-at faults for implementations produced by an automatic FSM synthesis system is achieved.

Table 6 - Single transition fault coverage by stuck-at fault test set

machine	Single transition faults		
	Detected	Total	Coverage
dk14	147	336	43.8%
dk15	31	96	32.3%
dk16	1676	2808	59.7%
dk17	98	224	43.8%
dk27	61	84	72.6%
dk512	295	420	70.2%
ex7	29	288	10.1%
planet	3381	5405	62.6%
opus	124	232	53.4%
bbara	97	252	38.5%

Acknowledgement - The authors would like to thank V.D. Agrawal and A. Dahbura for the stimulating discussion on sequential machine testing.

References

1. W. T. Cheng, "The BACK Algorithm for Sequential Test Generation," *Proc. Int. Conf. Computer Design (ICCD-88)*, pp. 66-69, October 1988.
2. F. C. Hennie, "Fault-Detecting Experiments for Sequential Circuits ," *Proc. 5th Ann. Symp. on Switching Circuit Theory and Logical Design*, pp. 95-110, November 1964.
3. K. T. Cheng and V. D. Agrawal, "A Partial Scan Method for Sequential Circuits with Feedback," *IEEE Trans. on Computers*, vol. 39, pp. 544-548, April 1990.
4. K. T. Cheng and V. D. Agrawal, "Design of Sequential Machines for Efficient Test Generation," *Proc. Int'l Conf. Computer-Aided Design*, pp. 358-361, November 1989.
5. K.-T. Cheng and J.-Y. Jou, "A Single Transtion Fault Model for Sequntial Machines," *in preparation.*.
6. Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, New York, 1978. Second Edition.
7. K. Sabnani and A. Dahbura, "A Protocol Test Generation Procedure," *Computer Networks*, vol. 15, pp. 285-297, 1988.
8. E. Hsieh, "Checking Experiments for Sequential Machines," *IEEE Trans. Computers*, vol. C-20, pp. 1152-1166, Oct. 1971.