FAULT DETECTION: STATE BEHAVIOR, MACHINE DECOMPOSITION AND ECONOMICS

Dissertation for the Degree of Ph. D.

MICHIGAN STATE UNIVERSITY
LARRY LEE GROVER

1976

This is to certify that the

thesis entitled

FAULT DETECTION: STATE BEHAVIOR,
MACHINE DECOMPOSITION AND ECONOMICS

presented by

Larry Lee Grover

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Computer Science

Carl V. Page
Major professor

Date August 9, 1976

O-7639

LL PHD001

ABSTRACT

FAULT DETECTION: STATE BEHAVIOR,
MACHINE DECOMPOSITION AND ECONOMICS

By

Larry Lee Grover

Conducting checking experiments to detect faults in sequential machines inherently requires the use of specific sub-experiments. In this dissertation, these sub-experiments, or state identification sequences, are used to verify the (indirectly observable) transition function. Further, an algorithm for determining state identification sequences is constructed from a rigorous treatment of state distinguishability and equivalence. This algorithm detects the shortest state identification sequence for any state of a machine for which state identification sequences exist.

For machines lacking the desired behavior, procedures are derived such that these machines can be designed to possess the required behavior. The notion of inaccessible states is given serious attention to provide a means of detecting faults which increase the number of states in these machines. The number of states may be increased by

faults creating access to the inaccessible states. Design procedures are given to make possible the detection of this type of fault.

A machine notation is introduced to detect and eliminate redundancies in checking experiments by determining when sub-experiments overlap. This notation and reduction technique is used in an algorithm to detect faults which can occur in a sequential machine. In this algorithm, the use of state identification sequences which satisfy specific conditions assists in making the checking experiments as concise as possible.

Machines which can be decomposed into serial or parallel connections of submachines possess certain structural properties which are beneficial to fault detection. These properties and their relationship to fault detection are presented and investigated to yield a fault detection method for decomposable machines. As a result of this method, both a savings in the amount of work required and other benefits are realized.

The cost (in a global sense) of fault detection is presented. Through a partial ordering of events which are encountered in fault detection, a cost model results. The use of this cost model establishes the overall fault detection program which has minimal cost.

FAULT DETECTION: STATE BEHAVIOR, MACHINE DECOMPOSITION AND ECONOMICS

By

Larry Lee Grover

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1976

ACKNOWLEDGMENTS

My special thanks to Dr. Carl V. Page whose enthusiastic guidance made this work pleasurable. I have enjoyed the opportunity to freely exchange ideas with a person of such keen ability. My thanks also to Dr. Richard C. Dubes, Dr. Harry G. Hedges, Dr. Richard J. Reid, Dr. James H. Stapleton and Dr. Bernhard Weinberg for their service in reviewing and guiding my work.

TABLE OF CONTENTS

Chapter		Page
1.	INTRODUCTION	1
	1.1 SEQUENTIAL MACHINE NOTATION	2
	1.2 THE PROBLEM OF FAULT DETECTION	5
	1.3 BACKGROUND AND HISTORY	6
	1.4 DISSERTATION ORGANIZATION	8
2.	FORMALIZATION OF FAULT DETECTION PROPERTIES OF SEQUENTIAL MACHINES	11
	2.1 STATE EQUIVALENCE AND DISTINGUISHABILITY	12
	2.2 STATE BEHAVIOR SEQUENCES	14
	2.3 PROPERTIES REGARDING STATE IDENTIFICATION SEQUENCES	18
	2.4 DESIGN PROVISIONS TO ENSURE A COMPLETE IS SET	24
	2.5 CONCERNING INACCESSIBLE STATES	28
	2.6 INITIALIZING THE MACHINE	31
	2.7 CHAPTER SUMMARY	31
3.	HOW TO MAKE A MACHINE MORE EASILY TESTABLE BY DESIGN	33
	3.1 DETERMINING STATE IDENTIFICATION SEQUENCES	33
	3.1.1 STATE IDENTIFICATION SEQUENCE ALGORITHM	36
	3.2 ALGORITHM FOR ADDITIONAL OUTPUTS	41
	3.3 ADDITIONAL INPUT ALGORITHM	46
	3.4 INACCESSIBLE STATE ASSIGNMENT ALGORITHM	47
	3.5 CHAPTER SUMMARY	52
4.	AN ALGORITHM FOR DETECTING FAULTS IN SEQUENTIAL MACHINES	54
	4.1 STRONGLY CONNECTED AND REDUCED MACHINES	55
	4.2 WORKING FORM NOTATION	57
	4.2.1 WORKING FORM ALGORITHM	58
	4.3 TRANSFER SEQUENCES	59
	4.3.1 TRANSFER SEQUENCE MATRIX ALGORITHM	61
	4.4 SUB-EXPERIMENT OVERLAPPING AND CONCATENATION	64
	4.5 FAULT DETECTION ALGORITHM	68

Chapter Page

4.6 EXAMPLES 70
4.7 BOUNDS ON EXPERIMENT LENGTH 75
4.8 CHAPTER SUMMARY 77

5. FAULT DETECTION THROUGH ABSTRACT REPRESENTATIONS . 79

5.1 SERIAL AND PARALLEL DECOMPOSITION 79
5.2 PARALLEL DECOMPOSITION AND FAULT DETECTION . 91
5.3 SERIAL DECOMPOSITION AND FAULT DETECTION . . 102
5.4 THE BENEFITS OF MACHINE DECOMPOSITION 111
5.5 CHAPTER SUMMARY 115

6. ECONOMICS OF CHECKING EXPERIMENTS 117

6.1 EXPERIMENT COST 118
6.1.1 MACHINE DESIGN AND PREPARATION FOR TESTING 120
6.1.2 EXPERIMENT DESIGN 121
6.1.3 PERFORMING THE EXPERIMENT 121
6.2 COST MODEL 123
6.3 MULTIPLE USE OF A CHECKING EXPERIMENT 135
6.4 CHAPTER SUMMARY 137

7. CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK 139

7.1 CONCLUSIONS 139
7.2 SUGGESTIONS FOR FUTURE WORK 141

BIBLIOGRAPHY 143

CHAPTER 1

INTRODUCTION

This dissertation is concerned with the detection of faults in sequential machines.

Definition 1.1

A sequential machine fault is any alteration which permanently changes the machine's logical operation.

Thus, failures which affect shapes of pulses, delay times or physical amplitudes, in the machine, but do not change its logical description, will not be considered. Also excluded are failures due to power sources and clock pulses.

Like other theories in science and engineering, the fault detecting theory presented here is mathematically based on the phenomena which characterize the subject. Thus, the theory and algorithms herein developed, apply to any entity which can be represented as a sequential machine.

The remainder of this chapter is used to define the sequential machine mathematical structure used in this

dissertation, define the problem, give its history and state the organization of this dissertation.

1.1 Sequential Machine Notation

The Mealy sequential machine model will be used in this dissertation (37).

Definition 1.2

A Mealy type sequential machine, M, is defined as a quintuple

$$M = (S, I, O, \eta, \rho)$$

where 1) S is a finite nonempty set of states;
2) I is a finite nonempty set of inputs;
3) O is a finite nonempty set of outputs;
4) η: S x I $\rightarrow$ S is called a transition function;
5) ρ: S x I $\rightarrow$ O is called the output function.

Let a single input symbol be denoted $x_i \in I$. A concatenation of k input symbols is denoted $\bar{x}_k$ and $(\bar{x}_k)_j$ implies the j^{th} string of k inputs. A single output response is denoted as $r_n \in O$, while $\bar{r}_k$ denotes a string of k responses. A transfer sequence from state s_i to s_j is a sequence of inputs denoted as $\tau(s_i, s_j)$. The next state for a machine in state s_i given the input x_j, is $\eta(s_i, x_j)$, while the output response is $\rho(s_i, x_j)$. The concatenation

of responses for a machine in state s_i given the input string $\bar{x}_k$, is $\bar{\rho}(s_i, \bar{x}_k)$.

This dissertation is concerned only with synchronous sequential machines. Sequential machines are classified as synchronous or asynchronous depending on whether or not the machine is operating under the control of clock pulses. In synchronous operation at most one internal state transition occurs in conjunction with the controling clock pulse. The output pulses are also in synchronism with the clock pulses.

Let x(t) and s(t) be the input and state of a synchronous sequential machine at time t. The state at time $t' = t+\Delta t$ is

$$s(t') = \eta_c(s(t), x(t))c + \eta_{\bar{c}}(s(t), x(t))(1-c),$$

where c is the synchronizing clock pulse train and

$$c = \begin{cases} 1, \text{ clock pulse present} \\ 0, \text{ otherwise.} \end{cases}$$

The logical description of a synchronous sequential machine is typically given showing the c=1 condition. When c=0, all states in the machine are stable for every $x_1 \in I$. Thus,

$$\eta_{\bar{c}}(s(t), x(t)) = s(t).$$

Suppose $\eta_{\bar{c}}(s_j, x_m)$ is faulty such that

$$\eta_{\bar{c}}(s_j, x_m) = s_f,$$

and

$$\eta_{\bar{c}}(s_f, x_m) = s_f.$$

Then the machine in any state s_l such that

$$\eta_c(s_l, x_m) = s_j,$$

will have s_j as a transient state on its way to s_f when the clock pulse returns to zero. Thus,

$$\eta_c(s_l, x_m) = s_f$$

because when the machine is exposed to the next clock pulse it will be in s_f. Hence, faults in the $\eta_{\bar{c}}$ portion of η appear as faults in η_c. Consequently, this dissertation is only concerned with the machine's description when the clock is present, following the tradition established in previous fault detection and automata theory. Therefore, any references to η will mean η_c in the remainder of this work.

The methods given in this dissertation can be extended to asynchronous machines, Friedman and Menon (15).

1.2 The Problem of Fault Detection

We shall assume that only the output function, ρ, is observable; the transition function, η, is not. Consequently all information concerning a machine's behavior must come through the output. It is therefore necessary to conduct experiments using only the terminals of a machine to determine its behavior. The experiments are derived such that inputs are applied and outputs observed to determine correspondence between the machine and its flow table description. The entire experiment is referred to as a checking experiment.

Definition 1.3

A checking experiment (CE) is a sequence of input symbols and their related output responses for the purpose of determining whether or not a sequential machine is functioning correctly.

A checking experiment may be either preset, in which the output response does not determine the next input symbol, or adaptive, in which the next input is determined from the present and past outputs.

Solving the fault detection problem is then a matter of devising a theory and algorithms by which checking experiments can be constructed such that they are concise and complete in detecting faults.

1.3 Background and History

The first work in sequential machine fault detection was by Moore (39). He showed that any sequential machine with n states, p input symbols and q output symbols, can be distinguished, up to equivalence, from all other (n, p, q)-machines by a simple checking experiment. The approach is to construct a machine which is the direct sum of all possible (n, p, q)-machines and then find an experiment which will distinguish the object machine from all others.

Because Moore's procedure requires prohibitively long experiments, Poage and McCluskey (42), Roth et al. (44) and Seshu and Freeman (46) greatly restrict the number of machines in the direct sum to reduce experiment length. This is accomplished by selecting a fault or small set of faults to determine the machines for the direct sum. This, however, restricts the diagnostic capability of the experiment. This approach is referred to as the "circuit-testing approach".

Hennie (23) seeks to devise experiments which identify a correct machine. The application of his experiment to any other machine of the same number of states or less produces a different response and hence, the experiment is a checking experiment.

Hennie's work on machine identification in fault detection is the basis for efforts by Boute (4), Farmer (13), Gönenc (19), Hsieh (27), Kime (31) and Kohavi et al. (34).

The chief result of their work has been to refine Hennie's procedures to make the resulting checking experiments more concise. The areas of refinement have been the removal of redundancy in the experiment and reduction in the length of certain input sequences used in Hennie's procedures.

Recently, work has been directed toward restricting the type of faults allowed e.g., Boute (5), Boute et al. (6), and Friedman et al. (14). Certain constraints are placed on the type of machine and internal logic interaction (states and output logic cannot be faulty simultaneously). The restrictions make techniques devised in this manner less potent than the machine identification approach but more general than the circuit-testing approach.

Aside from these works, fault detection has received much attention as evidenced by the literature. Five issues of the IEEE Transactions on Computers* have been devoted exclusively to Fault-Tolerant Computing of which fault detection is a major area Carter (8). Five symposia on Fault-Tolerant Computing** have been conducted and at least two texts (9), (15) dealing with fault detection and diagnosis in digital circuits have been written. Other texts which consider machine experiments and fault detection in their scope also exist (17), (24), (32) and

---

* Vol. C-25, July 1976, vol. C-24, May 1975, vol. C-23, July 1974, vol. C-22, March 1973, vol. C-20, Nov. 1971.
** IEEE International Symposium on Fault-Tolerant Computing, June 1975, June 1974, June 1973, June 1972, March 1971.

(48). Research in fault detection at Michigan State University has resulted in two publications, (10) and (29).

1.4 Dissertation Organization

The following algorithm exhibits both the organization of this dissertation and the use of this thesis in fulfilling one's needs concerning fault detection. In the flowchart following the algorithm, the numbers in the decision operations refer to steps in the algorithm and the numbers in the process operations refer to sections of the dissertation:

1. If machine decomposition or partitioning is a strong consideration, go to 15; otherwise continue.
2. Refer to Sections 3.1 and 3.1.1 to determine whether or not a state identification sequence (Def. 2.6) exists for each state of the machine.
3. If the machine has a state identification sequence for each of its states, go to 8; otherwise continue.
4. If the machine is already constructed, go to 7; otherwise continue.
5. Refer to Sections 2.4, 3.2 and 3.3 for theory and design procedures which ensure an identification sequence for each state.
6. Consider adding inputs, outputs and testpoints and determine the machines for all three and go to 8.
7. If the machine can be altered for fault detection

through redesign, go to 5; otherwise go to 15.

8. Given that the machine has n states and v internal state variables which can have any of r values, if $n=r^v$, go to 11; otherwise continue.

9. If faults which can cause the r^v-n inaccessible states to be entered are not of concern, go to 11; otherwise continue.

10. Refer to Sections 2.5 and 3.4 for the theory and design procedures to ensure that faults causing entry to inaccessible states are detected.

11. Refer to Sections 2.6 and 4.1-4.5 to determine a checking experiment for the machine using state identification sequences.

12. Determine the costs associated with fault detection (Section 6.1).

13. Refer to Sections 6.1-6.3 for economical analysis.

14. Construct cost model and find the best set of admissible costs and go to 22.

15. Refer to Section 5.1 and perform machine decomposition (Hartmanis and Stearns (22)).

16. If a nontrivial decomposition exists, go to 18; otherwise continue.

17. Refer to Section 1.3 for a guide to methods of deriving checking experiments using other than state identification sequences and go to 12.

18. Refer to Sections 3.1 and 3.1.1 to determine whether or not each submachine has a complete set of state

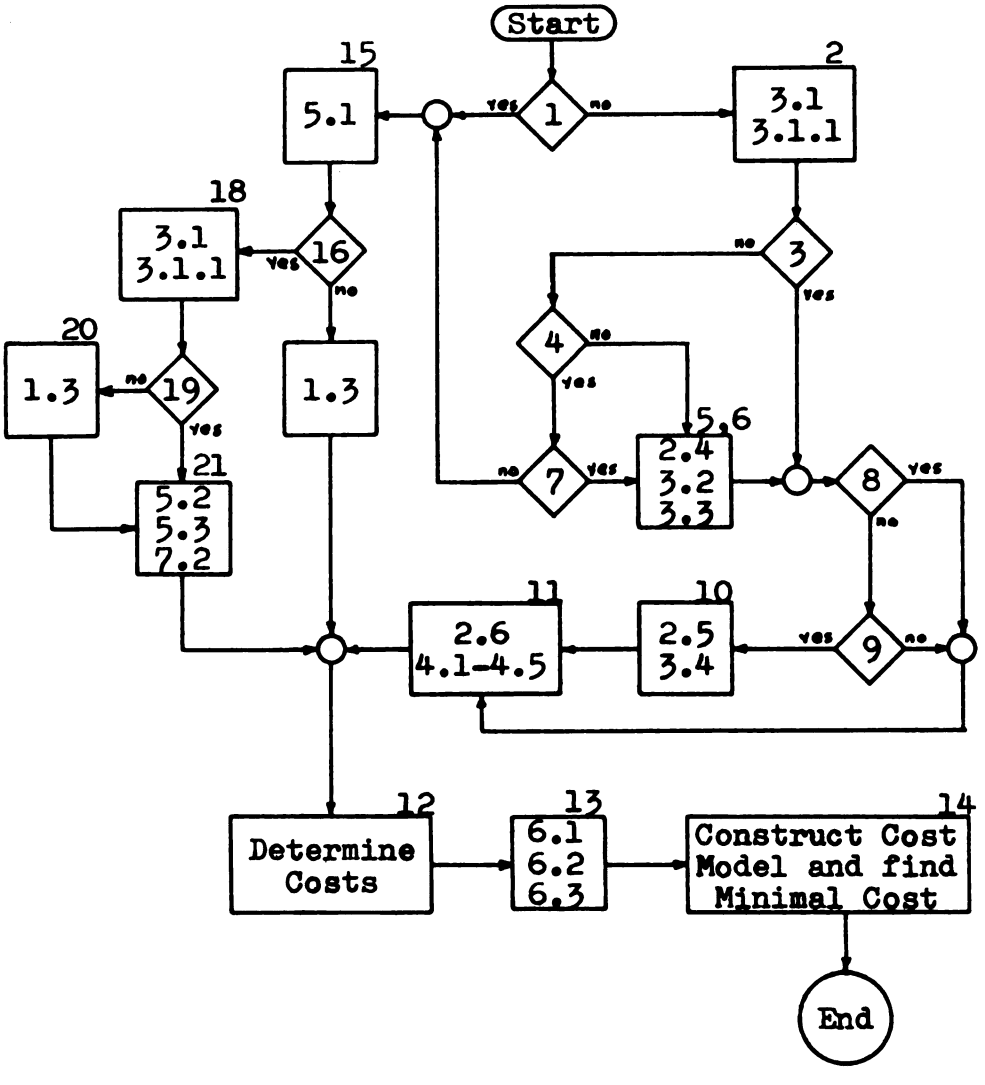
identification sequences.

19. If all submachines have a complete set of state identification sequences, go to 21; otherwise continue.

20. Refer to Section 1.3 for a guide to methods of deriving checking experiments for those machines not having a complete set of state identification sequences.

21. Refer to Sections 5.2, 5.3 and 7.2 for theory relating faults to machine decomposition and go to 12.

22. End.



CHAPTER 2

FORMALIZATION OF FAULT DETECTION PROPERTIES OF SEQUENTIAL MACHINES

The transition function, η, is not directly observeable by terminal measurements. However, the manner in which a machine responds to an input sequence does reveal information about its states. It is this state behavior that is of concern in this chapter.

Definition 2.1

An I/O sequence is an input sequence (I) and its related output sequence (O).

From Definition 1.3, a checking experiment (CE) is an I/O sequence which serves to detect faults in the machine. The particular sequences which are defined in this chapter may eventually be used as part of a checking experiment. Hence, these sequences will constitute a sub-experiment for a checking experiment.

Definition 2.2

Any I/O sequence incorporated in the construction of

a CE is a sub-experiment.

2.1 State Equivalence and Distinguishability

To devise a sequence (experiment) which will determine the behavior of the machine initially in a particular state, it is necessary to find some input sequence which will distinguish one state from the others. This becomes impossible if two or more states of the machine are equivalent.

Definition 2.3

State s_i and s_j of a sequential machine are said to be equivalent, if

$$\bar{\rho}(s_i, (\bar{x}_m)_p) = \bar{\rho}(s_j, (\bar{x}_m)_p)$$

for all m and p, otherwise they are distinguishable.

Definition 2.4

State s_i and s_j of a sequential machine are said to be k-equivalent, if

$$\bar{\rho}(s_i, (\bar{x}_m)_p) = \bar{\rho}(s_j, (\bar{x}_m)_p)$$

for all p, and $m \leq k$, otherwise they are k-distinguishable.

The following Lemma by Gill (17) says that if some

input sequence of length k, say $\bar{x}_k$, will distinguish states s_i and s_j, then any sequence $\bar{x}_k\bar{x}_m$, $m \geq 0$, will also.

Lemma 2.1:

(a) If two states are k-equivalent, then they are j-equivalent for every $j \leq k$. (b) If two states are k-distinguishable, then they are j-distinguishable for every $j \geq k$.

The search for input sequences which distinguish between states begins with the shortest possible input sequence and progressively concatenates inputs until the sequence is found. The notion of states which merge is also useful in determining rules for a process which is to be defined later.

Definition 2.5

State s_i and s_j of a sequential machine are said to merge, if for some input symbol, x_l,

$$\eta(s_i, x_l) = \eta(s_j, x_l).$$

Theorem 2.1:

Suppose g states of a sequential machine respond identically for some input sequence $(\bar{x}_m)_j$. Then, if these g states all merge given $(\bar{x}_m)_j$, they are equivalent with

respect to every input sequence having $(\bar{x}_m)_j$ as a prefix.

This theorem follows immediately from Gill's lemma so the proof is omitted. Denote this type of equivalence as $(\bar{x}_m)_j$-prefix equivalence. These notions of equivalence and distinguishability are important to the rules which govern the process used to find state identification sequences.

2.2 State Behavior Sequences

This section is devoted to defining and discussing sequences (experiments) which determine state behavior.

Definition 2.6

A state identification sequence (IS) for s_i is an I/O sequence $(\bar{x}_{IS}/\bar{\rho}(s_i, \bar{x}_{IS}))$ such that

$$\bar{\rho}(s_i, \bar{x}_{IS}) \neq \bar{\rho}(s_j, \bar{x}_{IS})$$

for all $j \neq i$.

An identification sequence for state Q of a machine is denoted IS_Q. If a machine possesses an identification sequence for each of its states, then it is said to have a complete IS set.

Definition 2.7

A distinguishing sequence (DS) is an input sequence ($\bar{x}_{DS}$) such that for an n-state sequential machine,

$$\bar{\rho}(s_i, \bar{x}_{DS}) \neq \bar{\rho}(s_j, \bar{x}_{DS})$$

for all i and j, $j \neq i$.

Therefore, a distinguishing sequence is an input sequence which defines a state identification sequence for every state of the machine. Thus, it is possible to determine the initial state of the machine simply by observing the response of the distinguishing sequence.

Not every machine has a distinguishing sequence. Certainly, machines which have equivalent states do not. Also, not every machine has a complete IS set as will be shown in later examples. The set of machines which do have distinguishing sequences are properly contained in the set of machines which have complete IS sets.

Distinguishing sequences and state identification sequences can be used to verify that a machine was in a given state and if so, infer the behavior of the machine throughout the sequence. There exists sequences which, although they cannot determine the initial state, can decide the state of the machine at the end of the experiment.

Definition 2.8

A homing sequence (HS) is an input sequence ($\bar{x}_{HS}$), such that the final state of the machine can be determined uniquely from the machine's response to $\bar{x}_{HS}$ regardless of the initial state.

Definition 2.9

A synchronizing sequence (SS) is an input sequence ($\bar{x}_{SS}$) which has

$$\eta(s_i, \bar{x}_{SS}) = s_k,$$

where s_k is the same for all i.

Hence, by observing the response of the machine to a homing sequence, it is assured that its final state is known. However, no predictions about the state of the machine during the experiment can be made. It has been shown that every machine which does not contain equivalent states has a homing sequence (17). Every synchronizing or distinguishing sequence is a homing sequence. Not every machine has a synchronizing sequence. These sequences are useful for maneuvering the machine into a desired state.

Example 2.1

The following machine, M_0, displays the four types of sequences which have been defined.

	e	f
A	A,w	B,w
B	A,w	C,w
C	C,w	D,y
D	D,y	B,y

M_0

The shortest identification sequence for state D is

$$IS_D = e/y.$$

No other state can produce a response of y to the input e.

The application of the input sequence efef to M_0 will leave the machine in state B regardless of its initial state. Thus, efef constitutes a synchronizing sequence.

A distinguishing sequence for this machine is ff. The following shows the states and their responses to the distinguishing sequence.

	ff
A	ww
B	wy
C	yy
D	yw

Any distinguishing sequence is also a homing sequence and this one is too. Another homing sequence for this

machine is ef. The following shows the possible responses to ef and the final states which these responses define.

initial	ef	final
A	ww	B
B	ww	B
C	wy	D
D	yy	B

A complete IS set for M_0 is the following where only the shortest state identification sequences are given.

IS_A = ff/ww
IS_B = ff/wy
IS_C = ef/wy, fe/yy, ff/yy
IS_D = e/y

2.3 Properties Regarding State Identification Sequences

This section concerns properties of sequential machines related to state identification sequences. These properties have a major bearing on the development of this work. Given in Theorem 2.2 is a condition which ensures a complete IS set for a sequential machine.

Definition 2.10

A sequential machine is reduced if no two states are equivalent.

Lemma 2.2:

A complete IS set does not exist for any machine which is not reduced.

Proof:

Since the machine is not reduced, it has states which are equivalent. If two states are equivalent, then they respond identically for every input sequence and hence, neither has an identification sequence.

Using the results of Lemma 2.2 we now determine a specific condition for a machine flow table such that given this condition a machine is assured of having a complete IS set.

Theorem 2.2:

For an n-state reduced sequential machine, if

$$(\eta(s_i, x_1), \rho(s_i, x_1)) \neq (\eta(s_j, x_1), \rho(s_j, x_1))$$

for all i, $j \neq i$, and all $x_1 \in I$, then the machine has a complete IS set.

Proof:

We shall assume the negation of the consequent of the theorem and show that this contradicts the premise. Assume that there exists a state, say s_i, in a machine satisfying

the conditions of this theorem which responds the same as at least one other state, s_p (which depends on input sequence), for all input sequences. Then the machine does not have a complete IS set because s_i does not produce a unique response for any of all input sequences. Suppose further that n copies of the machine are available each one in a different state than the rest. Then m_i (the machine originally in s_i) will respond the same as at least one other machine for every input string. Now apply an input such that some machines are distinguished from the others. This is possible, for if it were not, all states would be equivalent and the machine was reduced to begin with. Now m_i is contained in a proper subset of the original n machines which were not distinguished from m_i. All machines in this subset are in different states because if any were in the same state, some input must have caused these states to merge into the same state and give the same output which is contrary to premise. Now apply an input sequence to distinguish between m_i and some other machine. This is possible because in a reduced machine all states are pair-wise distinguishable. Delete the machines so distinguished from m_i. Continue this process until only two machines remain, m_i and some other machine, which have not been distinguished between by all previous input strings. These two machines are in different states due to previous arguments. Now apply the input sequence which distinguishes the two states and m_i has been found which is a contradiction to the assumption.

On the other hand, there exists a situation which prevents a state from possessing a state identification sequence. This is evident in Theorem 2.3.

Theorem 2.3:

If state s_i in an n-state sequential machine has for each member of its input alphabet, x_1,

$$\eta(s_i, x_1), \rho(s_i, x_1) = \eta(s_j, x_1), \rho(s_j, x_1)$$

for some $j \neq i$, s_i does not have an identification sequence.

Proof:

For every possible input symbol s_i is $(\bar{x}_m)_k$-prefix equivalent to another state. Therefore, no concatenation of input symbols exists which distinguishes s_i from any other state.

To illustrate the situation presented in Theorem 2.3, consider the following machine M_1.

	0	1
A	C,0	D,1
B	C,0	D,0
C	B,1	D,0
D	A,0	A,1

M_1

State B of M_1 responds the same as A and goes to the same next state as A for any input sequence having the initial input "0". State B also responds the same as C and goes to the same next state as C for all input sequences having the initial input "1". Thus, B responds the same as some other state of M_1 for all input sequences.

Thus, there exists two situations concerning a machine's logical description which provide insight for design criterion. One situation to avoid (Theorem 2.3) and one to strive for (Theorem 2.2).

Since state identification sequences are to be used in verification of a machine's flow table, a procedure will be given for determining these sequences. It is therefore necessary to determine how far this procedure must be carried out in its search. Theorem 2.5 gives the bounds and Theorem 2.4 whose proof has been given by Gill (17) is useful. This same result (Theorem 2.4) has also been given by Salomaa (45) where he shows that the result of Theorem 2.4 is the best upper bound.

Theorem 2.4:

Two states in an n-state machine are equivalent if they are (n-1)-equivalent and distinguishable if they are (n-1)-distinguishable.

Theorem 2.5:

Given that a state of an n-state sequential machine

has an identification sequence, the length of the sequence need not be longer than k, where

$$k = (n-1)^2.$$

Proof:

Assume n copies of the n-state machine are available each in a different state and the machine in state s_i is denoted m_i. Since s_i has an identification sequence, there exists some input sequence for which m_i will respond uniquely. Theorem 2.4 assures that there exists an input sequence of length n-1 such that m_i is distinguished, pair-wise, from at least one other machine. Lemma 2.1 assures that once m_i is distinguished from another machine by some sequence, any inputs concatenated with this sequence will always distinguish the two machines.

Assume that no sequence of length n-1 distinguishes m_i from more than one other machine. Now m_i is contained in a set of n-1 machines and must be distinguished from the other members. Since s_i possesses an identification sequence, the first n-1 symbols of the identification sequence must leave m_i in a different state than the other members of the n-1 member indistinguished set. If not, m_i would be $(\bar{x}_k)_j$-prefix equivalent to another machine and hence, the sequence being constructed could not be an identification sequence.

Now considering all possible input sequences of

length n-1 concatenated with the previous portion of the identification sequence, m_i must be distinguished from at least one member of the indistinguished set. After application of the j^{th} correct sequence, m_i has been distinguished from at least j machines.

Continuing this process n-1 times will result in m_i being completely distinguished from all other machines. Therefore, the construction process has, under worst conditions, yielded an identification sequence for s_i, such that the maximum length of the sequence is

$$k = \sum_{i=1}^{n-1} (n-1) = (n-1)^2.$$

The bound determined here does not imply that no sequence of length greater than $(n-1)^2$ is an identification sequence. It shows that if an identification sequence exists, one exists within the bound derived here.

2.4 Design Provisions to Ensure a Complete IS Set

Because a machine's output function, ρ, is directly observable, we investigate this function for design possibilities. A machine which does not have a complete IS set can be made to have one by providing additional outputs. The existence of such a design procedure follows from Theorem 2.6.

Theorem 2.6:

Any sequential machine having v internal r-ary variables can be designed to possess a complete IS set, by providing m additional output variables, where $m < v$.

Proof:

Given v internal variables and r possible values for each variable, there are r^v possible combinations of these variables. Hence, there are r^v possible distinct states for the machine and using v output variables, each state can be assigned a different combination of output components. Therefore, each state has an identification sequence.

The possibility of adding inputs in ensuring a complete IS set is evident in Theorem 2.7.

Theorem 2.7:

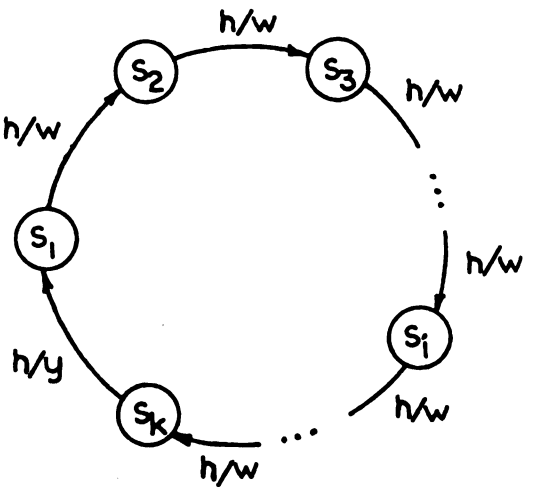
Any sequential machine can be made to have a complete IS set by the addition of only one member to its input alphabet

Proof:

Consider the following machine, M_h, which has only one input, h.

	h
s_1	s_2,w
s_2	s_3,w
s_3	s_4,w
	.
	.
	.
s_i	s_{i+1},w
	.
	.
	.
s_k	s_1,y

M_h



M_h State Diagram

Suppose M_h is initialized to s_i. After k-i inputs, $k > i$, the machine is in s_k and the response is k-i w's. The next input will produce a y response. Hence, initially in s_i, the machine will produce a y response on

the k-i+1 input. The following results are observed for an input sequence of k-1 h's.

State	$\overbrace{\text{hhh ... hhh}}^{k-1}$ ... hhh
s_1	www ... www ... www
s_2	www ... www ... wwy
s_3	www ... www ... wyw
.	
.	
.	
s_i	www ... wyw ... www
.	
.	
.	
s_k	yww ... www ... www
	↑ k-i+1

It is certainly evident that each state of this machine has an identification sequence. The state identification sequence for state s_j is

$$IS_{s_j} = \overbrace{\text{hh ... hh}}^{k-j+1}/\overbrace{\text{ww ... wy}}^{k-j+1},$$

where $j \neq 1$. The state identification sequence for s_1 is

$$IS_{s_1} = \overbrace{\text{hh ... hh}}^{k-1}/\overbrace{\text{ww ... ww}}^{k-1}.$$

It is also noted that the input string of k-1 h's is a distinguishing sequence. Thus, adding h as one additional member to the input alphabet and assigning it the entries contained in M_h yields a complete IS set for any sequential machine.

2.5 Concerning Inaccessible States

For every n-state sequential machine having v r-ary state variables where

$$r^{v-1} < n < r^{v},$$

there are r^{v}-n inaccessible states. Inaccessible states do not appear as next states in the n-state machine. Therefore, occurence of faults can produce paths to previously inaccessible states and consequently checking experiments should be able to detect these faults. It would be advantageous to assign to inaccessible states, next states and responses suitable for detecting entry into them.

If faults create paths to inaccessible states, any checking experiment which relies on verification of the flow table requires alteration of the machine structure to provide entry to the inaccessible states. Because faults in inaccessible states are possible also, there exists the possibility or certainty for each fault which causes entry to an inaccessible state, a fault in that inaccessible state which masks the fault causing entry.

Suppose that in the checking experiment, a fault causes $\eta(s_i, a)$ to become s_m (s_m inaccessible) instead of s_j. Now $\eta(s_m, b)$ is s_j and $\rho(s_m, b)$ is g. Since $\eta(s_j, b)$ is s_j and $\rho(s_j, b)$ is f, the fault causing entry into s_m would be detected by an input of b. If $\rho(s_m, b)$ is faulty and becomes f, the machine will be in the correct state and no fault is detected. If the checking experiment subjects state s_i to an a input only once, the fault is not detected by the checking experiment. It is necessary to verify the transitions in the inaccessible states as well. This can only be done if access, in some form, to the inaccessible states is available.

One way to solve this access problem would be to provide an additional member to the input alphabet which is used only during testing. This then would allow the previously inaccessible states to be entered as long as it was not faulty.

In assigning entries to the r^v-n states, a primary goal is that each state have an identification sequence to ensure a complete IS set. From Theorem 2.2, it is necessary to assign unique entries under every input for the inaccessible states. The next state entries for these states are states contained in the original n. The reasoning will become evident in ensuing arguments.

Theorem 2.8:

For an n-state, p-input, q-output sequential machine

which has v internal r-ary state variables, there are r^v-n possible distinct entries in the flow table available for each input symbol when $n > r^v/2$, using only the original n states.

Proof:

There are a maximum of nq possible distinct entries for each input symbol using the original n states. Since a maximum of n are used for an input, n(q-1) remain. Hence, n(q-1) must be greater than or equal to r^v-n in order to supply the unique entries which will satisfy Theorem 2.2. Thus,

$$n(q-1) \geq r^v-n$$

or

$$q \geq r^v/n.$$

For nontrivial sequential machines,

$$q \geq 2 > r^v/n.$$

Theorem 2.8 guarantees that sufficient distinct entries are available. It remains to find and assign these entries. The procedure which will accomplish the desired assignments is given in Chapter 3.

2.6 Initializing the Machine

Given a sequential machine for testing, there may be no way of knowing its present state. Since only terminal experiments are allowed here, it is necessary to force the machine into a known state by terminal experiments. Maneuvering the machine into a known state is referred to as initialization. Once initialized, the future behavior of the machine can be predicted. Synchronizing sequences and adaptive homing techniques are useful for initialization and their derivation is given in Gill (17).

2.7 Chapter Summary

The necessity of experimentally observing the transition function has resulted in a study of sequential machine state behavior. The primary concern in this chapter was to define and detect particular state behavior which could identify either the initial or final state of a machine with respect to the application of an input sequence.

The means by which states could be identified has stemmed from the notions of state equivalence and distinguishability. These notions have been formally developed and used extensively in the derivation of algorithms (Section 3.1.1) which generate the desired input sequences that identify states through output observations.

This chapter presents particular machine characteristics which are useful in machine design as applied to

fault detection. A discussion of unused or inaccessible states is also given in Section 2.5.

CHAPTER 3

HOW TO MAKE A MACHINE MORE EASILY TESTABLE BY DESIGN

Not all sequential machines possess a complete state identification sequence set or a distinguishing sequence. Consequently one must rely on testing techniques introduced by Hennie (23), which result in checking experiments of great length, unless the machine can be designed to yield additional information about its states and transition function. This chapter will consider the design of machines which are more readily testable. In particular, faults which increase the number of accessible states can be made testable by design. The design criteria presented in this chapter can be used in conjunction with usual design practices resulting in a single design phase.

3.1 Determining State Identification Sequences

Although state identification sequences are discussed in the literature, efficient means for computing them are not, although there may be proprietary programs to do this. One way to solve the problem of finding state

identification sequences for a finite state sequential machine would be to determine if the machine, as an encoder, encodes (state, input) information into output sequences which are uniquely decodable (Abramson (2)). Thus,

$$(s_i, \bar{x}_{IS_{s_i}}) \rightarrow \bar{\rho}(s_i, \bar{x}_{IS_{s_i}})$$

would represent an encoding which is uniquely decodable.

In this dissertation we present a means of determining state identification sequences for any sequential machine by using the successor tree. Gill (17) explains the successor tree and how he uses it to find distinguishing and homing sequences.

The successor tree is, in itself, infinite and of no practical use. To make the successor tree useful in the derivation of machine experiments, termination rules must be established such that branches in the tree are terminated when determinated useless or the tree growth itself is to be halted for some specified reason.

In the process of determining possible state identification sequences, particular input sequences may be found which will not assist in finding an IS for a state not having one already. Once found, these input sequences (tree paths) are omitted from further use. An input sequence which is retained offers potential value in

determining the desired identification sequences and is referred to as a potential input sequence.

The procedure to be developed here does not use the tree structure in the graphical sense but, uses a compressed representation of the tree to facilitate the tree searching. This representation consists of an array (G) acting as successor tree levels. Each row of G corresponds to a different state in the machine. The columns of G correspond to input sequences (tree paths to the particular tree level G represents). Given that the machine's state set is

$$S = \{s_1, s_2, \ldots, s_n\},$$

assign s_i to the i^{th} row and $(\bar{x}_m)_j$ to the j^{th} column. Now the entries (tree nodes for the level) in G are

$$(g, h)_{ij} = g_{ij}, h_{ij} = \eta(s_i, (\bar{x}_m)_j), \bar{\rho}(s_i, (\bar{x}_m)_j).$$

The array at each successive tree level is referred to as a generation. The zeroth-generation is simply the original flow table. The input sequences for the next generation are determined by concatenating all members of the input alphabet with the prior generation's potential input sequences. For the prior generation input $(\bar{x}_m)_j$, the new generation input is $(\bar{x}_m)_j x_k$, $x_k \in I$. Let

$$(\bar{x}_m)_j x_k = (\bar{x}_{m+1})_p$$

for some p, then

$$g_{ip} = \eta(g_{ij}, x_k)$$

$$h_{ip} = h_{ij}\,\rho(g_{ij}, x_k)$$

and each generation is computed using only η, ρ and the most recent generation.

3.1.1 State Identification Sequence Algorithm

A partition on the machine's state set is a grouping of all states into disjoint subsets called blocks. A partition in which two states s_i and s_j are in the same block, if and only if

$$\bar{\rho}(s_i, (\bar{x}_m)_k) = \bar{\rho}(s_j, (\bar{x}_m)_k)$$

i.e.,

$$h_{ik} = h_{jk}$$

is said to be the partition induced by $(\bar{x}_m)_k$ and denoted $\pi((\bar{x}_m)_k)$. A partition in which two states s_i and s_j are in the same block, if and only if

$$\eta(s_i, (\bar{x}_m)_k), \bar{\rho}(s_i, (\bar{x}_m)_k) = \eta(s_j, (\bar{x}_m)_k), \bar{\rho}(s_j, (\bar{x}_m)_k)$$

i.e.,

$$(g, h)_{ik} = (g, h)_{jk}$$

is denoted $\omega((\bar{x}_m)_k)$. The state identification sequence algorithm follows.

1. The machine flow table is the zeroth-generation. Set k equal to zero.
2. For each $(\bar{x}_m)_j$ of the present generation, determine $\pi((\bar{x}_m)_j)$ and $\omega((\bar{x}_m)_j)$. Increment k by 1.
3. If any $\pi((\bar{x}_m)_j)$ contains a singleton block, then that state (s_i) in the block has a state identification sequence which is $(\bar{x}_m)_j/\bar{\rho}(s_i, (\bar{x}_m)_j)$. Indicate that s_i has a state identification sequence by a \$ and also the sequence.
4. If all states have been found to have state identification sequences, terminate with success; otherwise continue.
5. If all states which do not have state identification sequences are contained in nonsingleton blocks of $\omega(\bar{x}_m)_j)$, then place a star over column $(\bar{x}_m)_j$ indicating that it is of no further use.
6. If all inputs (columns) have been marked with a star, terminate without a complete IS set or no state

identification sequences at all; otherwise continue.

7. If $k = (n-1)^2$, terminate without a complete IS set; otherwise continue.

8. Construct the next generation i.e.,

$$g_{ip} = \eta(g_{ij}, x_k)$$

$$h_{ip} = h_{ij}\,\rho(g_{ij}, x_k),$$

for all $x_k \in I$ and potential $(\bar{x}_m)_j$ of the previous generation and go to step 2.

Example 3.1

This example illustrates the algorithm using machine M_2 which is defined by G_0.

G_0

	$\overset{*}{0}$	1
A	A,0	B,0
B	A,0	C,0
C	A,0	D,0
\$D	A,1	D,0

G_1

	$\overset{*}{10}$	11
A	A,00	C,00
B	A,00	D,00
C	A,01	D,00
D	A,01	D,00

G_2

	$\overset{*}{1}10$	$\overset{*}{1}11$
\$A	A,000	D,000
B	A,001	D,000
C	A,001	D,000
D	A,001	D,000

Since all remaining inputs have stars, the algorithm terminates with a partial IS set.

$$IS_A = 110/000$$
$$IS_D = 0/1$$

There are several aspects of this algorithm which are of interest to checking experiment theory.

Theorem 3.1:

The state identification sequence algorithm will determine a state identification sequence for each state of a sequential machine for which a state identification sequence exists.

Proof:

If a state does possess an identification sequence, then an input sequence exists for this state which will produce a unique response. Since an input's use is never terminated until it is known that all states without an identification sequence are $(\bar{x}_m)_j$-prefix equivalent to at

least one other state for each input (step 6), then the existence of such an input sequence ceases once the algorithm terminates.

Theorem 3.2:

If an identification sequence exists for a state of a sequential machine, the state identification sequence algorithm will determine the shortest such sequence.

Proof:

Prior to the construction of a new generation, the present generation is checked for all possible state identification sequences. Hence, prior to increasing an input's length, all possible state identification sequences are determined and noted.

Theorem 3.3:

The state identification sequence algorithm will terminate for any finite state machine.

Proof:

If the algorithm has not found an identification sequence within $(n-1)^2$ generations, none exist by Theorem 2.5 and the algorithm terminates in step 7.

3.2 Algorithm For Additional Outputs

The notion of adding outputs to machines to render more efficient fault detection has been considered e.g., Kohavi and Lavellee (33), Sheppard and Vranesic (47). This additional output assignment problem is considered in this dissertation and expanded such that when applied to sequential machines, complete IS sets are the result. The existence of such a design procedure follows from Theorem 2.6.

The process of providing additional outputs is, in effect, that of embedding the original machine in a new machine which provides additional state information. Let

$$(z_{01}, \ldots, z_{0j}) = \rho_0(s_i, x_l)$$

denote the original output vector and

$$z_m = \rho_m(s_i, x_l)$$

denote the m^{th} additional output. Then the output for the machine is a vector represented as

$$Z = (z_{01}, \ldots, z_{0j}, z_1, z_2, \ldots, z_m).$$

In order to simplify the notation, base 10 values of these vectors are used. Hence,

$$R = z_{01}b^{m+j-1} + \ldots + z_{0j}b^{m} + z_{1}b^{m-1} + \ldots + z_{m}$$

where b is the base of the number system for the internal variables written in base 10. Consider the following in base 2.

$$Z = (1, 0, 1)$$

$$R = 5$$

Now R is used as the output and the flow table entry is

$$s_i,(1, 0, 1) = s_i,5.$$

Theorems 2.2 and 2.3 suggest a design procedure for adding outputs to ensure that a complete IS set exists. An algorithm for this design process first considers whether or not the machine does have a complete IS set. In the event that it doesn't, an additional output is created to attempt completion of the IS set. The new output is specified such that it will make states previously $(\bar{x}_m)_j$-prefix equivalent no longer equivalent. Therefore, identical $\eta(s_i, x_1)$, $\rho(s_i, x_1)$ entries in the flow table for some input are made different by assigning the added output to make them different. Thus, these states become distinguishable at the earliest possible time.

The addition of outputs to the machine doesn't alter state identification sequences which existed prior to the addition. Existing output vector components are never changed. Thus, these components partition the machine's state set into blocks as they previously did. The added components simply break up nonsingleton blocks of $\omega((\bar{x}_m)_j)$ into smaller blocks. Additional outputs for states which possess identification sequences are free and may be used as don't care conditions (dc). In many cases, identification sequences for states which previously exhibited them become shorter. The added outputs create state information and Theorem 2.6 shows that if carried far enough, all states would have a state identification sequence of length one for every $x_1 \in I$. In example 3.2, state A results with a state identification sequence whose length is one less than without the output.

This algorithm makes use of $\omega((\bar{x}_m)_j)$ as defined in Section 3.1.1. Breaking up the blocks of $\omega((\bar{x}_m)_j)$, through additional outputs, results in state identification sequences for states which do not have them. The algorithm follows.

1. Initially set the index m to zero.
2. Apply the state identification sequence algorithm and increment m by 1.
3. Partition the states of the machine into two blocks ν_1 and ν_2 such that s_i is in ν_1, iff it has an IS.

4. If ν_2 is empty, stop; otherwise continue.

5. For each $x_1 \in I$, determine $\omega(x_1)$.

6. Construct an array (A), one row for each state in ν_2 and one column for each $x_1 \in I$. The entries for A are

$$a_{ij} = \left\{ s_j : s_j \neq s_i \text{ and } s_j \text{ is contained in the same block of } \omega(x_j) \text{ as } s_i \right\}.$$

7. For each s_i, assign z_m for all $x_1 \in I$ which has not had z_m assigned, as follows.

7a If $s_i \in \nu_1$, $\rho_m(s_i, x_1) = dc$.

7b If $s_i \in \nu_2$, $\rho_m(s_i, x_1) \neq \rho_m(s_j, x_1)$, for all $s_j \in a_{il}$.

8. Go to step 2.

The don't care situation allows the freedom of using this design procedure with any other state assignment criterion.

Example 3.2

Using M_2 from Example 3.1, steps 1, 2 and 3 have been completed previously with states B and C found not to have state identification sequences. Application of steps 4 and 5 result in the following.

	0	1
B	A,C	-
C	A,B	D

Steps 6 and 7 give rise to the new flow table M_2'.

	0	1
A	A,(0, 1)	B,(0, 1)
B	A,(0, 0)	C,(0, 1)
C	A,(0, 1)	D,(0, 0)
D	A,(1, 1)	D,(0, 1)

M_2'

In the simplified form this appears as the following.

	0	1
A	A,1	B,1
B	A,0	C,1
C	A,1	D,0
D	A,3	D,1

M_2'

Application of the state identification sequence algorithm yields the following complete IS set.

$IS_A = 10/10$

$IS_B = 0/0$

$IS_C = 1/0$

$IS_D = 0/3$

3.3 Additional Input Algorithm

To facilitate fault detection, an additional input has been considered by Kane and Yan (28), Murakami and Kinoshita (4). This notion of an additional input is expanded in this dissertation resulting in the capability of completing a machine's IS set. The state identification sequences generated are as short as possible. The possibility of an additional input in finding a means of making a machine more readily tested is evident in Theorem 2.7.

A primary goal in the derivation of checking experiments is to make the experiment as concise as possible. If a machine has a complete IS set, it is needless to add the additional input. If a machine has a partial IS set, then by a prescribed assignment of its state set to the state set of machine M_h (Section 2.4), a shorter set of identification sequences can be constructed for the machine. This is accomplished by choosing one of the states not having an identification sequence to be s_k. The next state without an identification sequence is assigned s_k as its next state, etc.. The procedure of adding one input to complete the IS set for a machine is the following.

1. Apply the state identification sequence algorithm to the machine.
2. If the machine has a complete IS set, terminate; otherwise continue.
3. If the machine has a partial IS set, continue; otherwise go to step 5.
4. For the states which do not have an identification sequence, arrange them such that they are s_k, s_{k-1}, s_{k-2}, etc. until all such states are covered. Now the states with identification sequences can be arranged in the remainder of M_h in any manner.
5. Append the input h to the original machine and assign the entries for this input using the cyclic pattern indicated in M_h.

The design of a machine using this procedure and the added input h will make any machine have a complete IS set. This is true even if the original machine is non-reduced or not strongly connected.

3.4 Inaccessible State Assignment Algorithm

Prior to the introduction of the material in this section, no state assignment procedures have been given in the literature which have attempted to design inaccessible states such that faults causing entry to these states can be detected. The inaccessible state assignment algorithm follows.

1. For a sequential machine with p input and q output symbols, construct an array (B) having pq rows and 2 columns. Each row corresponds to a different possible x_l/r_j combination where $x_l \in I$ and $r_j \in O$.
2. The entries in B are:

$$b_{(x_l/r_j)1} = \left\{ s_p : \text{for all } s_i,\ \eta(s_i, x_l) = s_p \text{ when } \rho(s_i, x_l) = r_j \right\}$$

$$b_{(x_l/r_j)2} = \left\{ s_p : s_p \in S \text{ and } s_p \notin b_{(x_l/r_j)1} \right\}.$$

3. For each inaccessible state (s_m), assign a unique $\eta(s_m, x_i)$, $\rho(s_m, x_i)$ for all $x_i \in I$, such that

$$(s_m, x_i) \in b_{(x_i/r_j)2}$$

and

$$\rho(s_m, x_i) = r_j.$$

When more than one choice is available in $b_{(x_l/r_j)2}$, any will suffice. However, certain design criteria may make one choice more desirable than another.

Theorem 3.4:

If the original n states of an n-state sequential machine have identification sequences, assignment of entries

to the r^v-n inaccessible states using the inaccessible state assignment algorithm results in a complete IS set for the machine.

Proof:

Either by Theorem 2.3 or the inaccessible state assignment algorithm every state s_i of the machine has some input symbol such that for all $i \neq j$,

$$\eta(s_i, x_m), \rho(s_i, x_m) \neq \eta(s_j, x_m), \rho(s_j, x_m).$$

Apply the particular x_m for s_i for which this is true. Now if the response is the same as some other state response, the next state must be different, and if the next state is the same, the response must be different which causes x_m to distinguish s_i from all s_j which have the same next state as s_i under x_m.

Since every state has as next states one of the original n and each of the original n has an input sequence which distinguishes it from the n-1 other accessible states, application of x_m concatenated with $IS_{\eta(s_i, x_m)}$ will distinguish s_i from all states. Hence, each of the r^v states has an identification sequence.

The only item remaining in designing a machine to ensure that it is capable of being tested for faults that cause paths to inaccessible states, is creating access to

the inaccessible states. In Section 2.4 the cyclic, single input, machine M_h was investigated and will serve as a model for creating the desired access.

Example 3.3

The following example illustrates this algorithm by using M_3.

	a	b
A	A,y	B,z
B	C,z	A,z
C	B,z	C,y

M_3

Steps 1 and 2 result in the following table.

	a	b
a/z	B,C	A
a/y	A	B,C
b/z	A,B	C
b/y	C	A,B

Step 3 yields the following machine.

	a	b
A	A,y	B,z
B	C,z	A,z
C	B,z	C,y
D	A,z	C,z

M_3'

Appending the single input h and its cyclic entries gives the following.

	a	b	h
A	A,y	B,z	B,z
B	C,z	A,z	C,z
C	B,z	C,y	D,z
D	A,z	C,z	A,y

M_3'

Application of the state identification sequence algorithm returns the complete IS set.

IS_A = a/y

IS_B = ab/zy, ba/zy, hb/zy

IS_C = b/y

IS_D = h/y

No criteria has been given for choosing the unique entries for inaccessible states. The question of which decision is best and how decisions affect the length of state identification sequences is left to future work.

3.5 Chapter Summary

The purpose of this chapter was to investigate how to make a machine more easily testable by design. It has been shown that machines can be designed using additional outputs or an additional input to generate a machine of higher dimensionality which has a complete IS set and a distinguishing sequence. The question of whether it is better to add one additional input or additional outputs must be answered by the designer. Adding one input requires checking n more transitions in an n-state machine. Additional outputs may require more than one terminal being added to the machine. The additional implementation which must be added in either case may also be a deciding factor.

The greatest depth that must be completed for the successor tree in the state identification sequence algorithm is $(n-1)^2$. The maximum depth required for other sequences discussed in this dissertation according to Gill (17), Ginsburg (18), Hibbard (25) and Kohavi (32) are:

SS: $(n-1)^2 n/2$;

HS: $n(n-1)/2$;

DS: $(n-1)n^n$.

In comparison to the distinguishing sequence, computation of a state identification sequence is shorter and requires less memory.

Since a state identification sequence is the shortest sequence which can be used to identify the state of a machine, the use of such sequences in checking experiments will result in checking experiments of shorter length than experiments using other sequences to identify states.

CHAPTER 4

AN ALGORITHM FOR DETECTING FAULTS IN SEQUENTIAL MACHINES

It has been shown that the present state of a machine can be verified if the machine has a complete IS set. Means of modifing machines which do not possess a complete IS set have been shown. The ability to verify the state of a machine is the basic tool used in an algorithm developed in this chapter. The machines to be considered in this fault detection algorithm must be completely specified, finite state and deterministic. This chapter also presents methods by which two classes of machines (non-strongly connected and nonreduced) can also be tested.

To make checking experiments produced by this algorithm efficient, a notational representation for the Mealy type sequential machine is introduced. This notation assists in eliminating redundancies in the checking experiment. Examples demonstrate the algorithm.

To establish apriori information regarding the length of checking experiments, bounds on experiment length are established.

4.1 Strongly Connected and Reduced Machines

In any fault detecting experiment which verifies the flow table transitions and associated responses, the experiment must be capable of maneuvering the machine into each state.

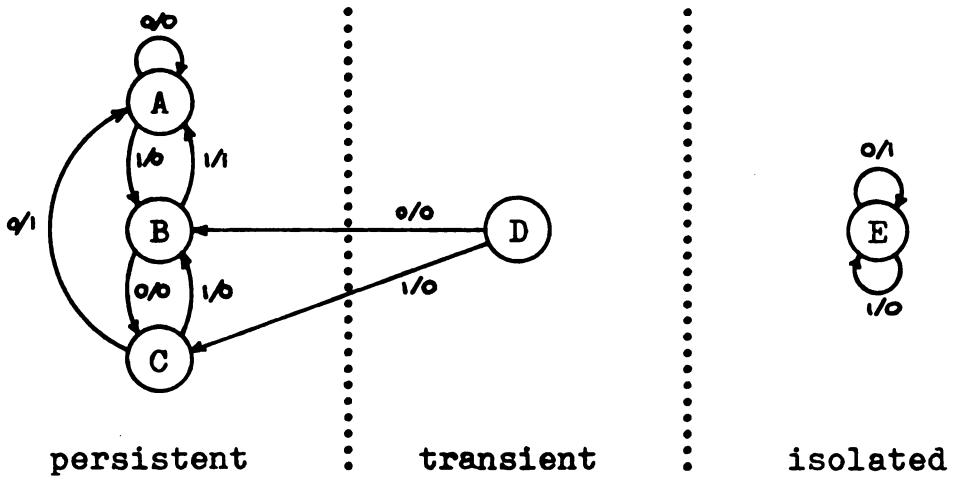
Definition 4.1

If for every pair of states s_i, s_j of a machine M there exists an input sequence which takes M from s_i to s_j, then M is said to be strongly connected.

Thus, non-strongly connected machines contain states which cannot be entered from other states. Non-strongly connected machines are those which have persistent, isolated or transient submachines. The following machine, M_4, exhibits these submachines most readily by means of a state diagram.

	0	1
A	A,0	B,0
B	C,0	A,1
C	A,1	B,0
D	B,0	C,0
E	E,1	E,0

M_4



State Diagram

Because this machine has inaccessible states, the arguments used for faults which increase the number of accessible states are applicable.

Machines which are not reduced (Definition 2.10) can be represented by machines having fewer states. Suppose a fault in a non-reduced sequential machine causes state $\eta(s_i, x_m)$ to be equivalent to the correct state. Then a fault exists and the machine performance is unchanged. We express this point in the following lemma which will not be proved.

Lemma 4.1:

In a non-reduced machine, a fault causing a transition to a different member of an equivalent set of states is in effect not a fault in that it does not change the behavior.

To derive a checking experiment for a non-reduced machine, reduce the machine flow table. Then use the state identification sequence found for all states, representing an equivalent set of states, as the identification sequence for each state of the set of states in which these representatives are contained.

4.2 Working Form Notation

A flow table representation called the "working form notation" is defined below. This notation is advantageous in fault detecting experiments because information about the machine is displayed in the experiment. The standard flow table notation, where columns represent inputs and rows represent states of the machine, uses

$$\eta(s_j, x_l), \rho(s_j, x_l)$$

as the row s_j column x_l entry. In working form notation

$$\eta(s_j, x_l), \rho(s_j, x_l)$$

is replaced by W_i^z, where W is $\eta(s_j, x_l)$, z is $\rho(s_j, x_l)$, and i is a unique pointer to each entry. An algorithm for going from flow table to working form notation is given in the following steps.

4.2.1 Working Form Algorithm

1. For each state of the machine, determine W_i^z for all x_1 in I, where:

$W = \eta(s_j, x_1)$;

$z = \rho(s_j, x_1)$;

i = c-1+(r-1)m+1, where r and c are row and column number and m is the maximum number of columns (inputs). The index i is unique for each entry.

2. Replace each $\eta(s_j, x_1)$, $\rho(s_j, x_1)$ by its corresponding W_i^z.

Example 4.1

To illustrate this algorithm, consider machine M_2 and its working form representation.

	0	1
A	A,0	B,0
B	A,0	C,0
C	A,0	D,0
D	A,1	D,0

M_2

	0	1
A	A_1^0	B_2^0
B	A_3^0	C_4^0
C	A_5^0	D_6^0
D	A_7^1	D_8^0

M_2 Working Form

From the following arbitrary experiment conducted on M_2, initially in state A, it is evident that the working

form representation of the experiment indicates at each step the input, the output, and the next state for the machine.

11011101111010110101	input
00000010000010000000000	output

$B_2^0C_4^0A_5^0B_2^0C_4^0D_6^0A_7^1B_2^0C_4^0D_6^0D_8^0A_7^1B_2^0C_4^0A_5^0B_2^0C_4^0A_5^0B_2^0A_3^0B_2^0$ Working Form

This notation is particulary useful for concatenating sub-experiments and determining when they overlap.

4.3 Transfer Sequences

In conducting machine experiments, the situation often arises where the machine is in one state and it is necessary to maneuver it into a different state. For small machines the transfer sequence is usually directly deducible from the flow table. However, for large machines, some necessary sequence or shortest sequence is not as easily found. Since conciseness is a goal in checking experiments, it will be necessary to have a procedure to find the shortest transfer sequence for a machine. The transition matrix is a useful tool for this purpose.

The transition matrix $T = [t_{ij}]$ for an n-state sequential machine is an n x n matrix. Given that the machine's state set is

$$S = \{s_1, s_2, \ldots, s_n\},$$

assignment of the j^{th} state (s_j) to the j^{th} row and column will make book-keeping an easier task. Entries in T are

$$t_{ij} = \{x_m: x_m \in I \text{ and } \eta(s_i, x_m) = s_j\}.$$

For the machine M_1 of Section 2.2, the following transition matrix is obtained where $\emptyset$ is the null set.

	A	B	C	D
A	$\emptyset$	$\emptyset$	0	1
B	$\emptyset$	$\emptyset$	0	1
C	$\emptyset$	0	$\emptyset$	1
D	0,1	$\emptyset$	$\emptyset$	$\emptyset$

The <u>transfer sequence matrix</u> is now constructed from the transition matrix. A transfer sequence is transitive. That is, if $\tau(s_i, s_j)$ and $\tau(s_j, s_m)$ exist, then $\tau(s_i, s_m)$ does also. The existence of $\tau(s_i, s_j)$ does not imply that of $\tau(s_j, s_i)$ and hence, a transfer sequence is not symmetric. The transfer sequence matrix is constructed by taking the transitive closure of the transition matrix using the following algorithm.

4.3.1 Transfer Sequence Matrix Algorithm

Prior to presenting the steps of the algorithm, some notation is considered. The transfer sequence matrix row i column j entry is referred to as t_{ij}. The length of a sequence is

$$L(\bar{x}_k) = k,$$

where k is the number of symbols in the sequence and

$$L(\emptyset) = \infty.$$

Since all members of t_{ij} are initially the same length and concatenations performed during the process do not cause any entries to have members of different lengths,

$$L(t_{ij}) = L((\bar{x}_m)_1 \in t_{ij}).$$

1. Construct the transition matrix for the machine and assign i the value 0.
2. Increment i by 1.
3. If i = n (number of states in machine), terminate; otherwise continue.
4. Assign j the value 0.
5. Increment j by 1 and assign m the value 1.
6. If j > n, go to step 2; otherwise continue.

7. If $j = i$ or $t_{ji} = \emptyset$, go to step 5; otherwise continue.

8. If $m > n$, go to step 5; otherwise continue.

9. Left concatenate t_{ji} with t_{im} and assign t_{jm} as follows.

$$t_{jm} = \begin{cases} t_{jm}, & \text{if } L(t_{ji}t_{im}) < L(t_{jm}) \\ (t_{jm} \cup t_{ji}t_{im}), & \text{if } L(t_{ji}t_{im}) = L(t_{jm}) \\ t_{ji}t_{im}, & \text{if } L(t_{ji}t_{im}) > L(t_{jm}). \end{cases}$$

10. Increment m by 1 and go to step 8.

The transfer sequence matrix algorithm is a result of applying the notion of Warshall's algorithm (50) to the sequential machine. The time complexity of transitive closure in the transfer sequence matrix algorithm for an n-state machine is no greater than n^3. Since only the shortest transfer sequences are retained during an iteration of the algorithm, the resulting matrix contains the shortest possible transfer sequences.

Example 4.2

Using this algorithm on M_1 yields the following results.

	A	B	C	D	
A	Ø	Ø	0	1	
B	Ø	Ø	0	1	
C	Ø	0	Ø	1	
D	0,1	Ø	00,10	01,11	i=1

	A	B	C	D	
A	Ø	Ø	0	1	
B	Ø	Ø	0	1	
C	Ø	0	00	1	
D	0,1	Ø	00,10	01,11	i=2

	A	B	C	D	
A	Ø	00	0	1	
B	Ø	00	0	1	
C	Ø	0	00	1	
D	0,1	000,100	00,10	01,11	i=3

	A	B	C	D	
A	10,11	00	0	1	
B	10,11	00	0	1	
C	10,11	0	00	1	
D	0,1	000,100	00,10	01,11	i=4

Thus, the transfer sequence matrix algorithm reveals

the shortest sequence of input symbols required to take the machine from one state to another. For machine M_1, the transfer sequence from state C to state A is either 10 or 11 and represents the shortest path. The process of finding the transfer sequence matrix is equivalent to that of finding the shortest path between any two nodes in a graph.

4.4 Sub-experiment Overlapping and Concatenation

The algorithm which will be presented in the next section is dependent on a "checking table", defined below. The checking table consists of 4 columns and np rows for an n-state, p-input sequential machine. Column 1 contains the indices for the working form representation in numerical order. The column labelled s_i,s_j has in row m the present state and the next state for the m^{th} index in the working form representation. The column labelled $\tau(s_i, s_j)$ has in row m the working form representation having the index m. The final column in the table is the working form representation of the identification sequences for the left entries in column s_i,s_j. The following illustrates the checking table.

	0	1
A	B_1^0	D_2^0
B	A_3^0	B_4^0
C	D_5^1	A_6^0
D	D_7^1	C_8^0

M_5

index	s_i, s_j	(s_i, s_j)	IS_{s_j}
1	A,B	B_1^0	$A_3^0 D_2^0 D_7^1$
2	A,D	D_2^0	$C_8^0 A_6^0 B_1^0 B_4^0 A_3^0$
3	B,A	A_3^0	$B_1^0 B_4^0 A_3^0$
4	B,B	B_4^0	$A_3^0 D_2^0 D_7^1$
5	C,D	D_5^1	$C_8^0 A_6^0 B_1^0 B_4^0 A_3^0$
6	C,A	A_6^0	$B_1^0 B_4^0 A_3^0$
7	D,D	D_7^1	$C_8^0 A_6^0 B_1^0 B_4^0 A_3^0$
8	D,C	C_8^0	$A_6^0 B_1^0 B_4^0 A_3^0$

Checking Table for M_5

Using the checking table, each $\tau(s_i, s_j)$ concatenated with IS_{s_j} constitutes a sub-experiment. Overlapping is commenced by assuming an initial index. The sub-experiment corresponding to this index begins the CE. The next

candidate for the CE is the sub-experiment whose index is the subscript of the second member in the CE. If all of the candidate's members coincide with the CE's members, when positioned over the CE member whose subscript nominated the candidate, the two overlap and any members of the candidates extending beyond the CE are retained. The next rightmost subscript is attempted to find a candidate and if its members do not coincide, the next rightmost subscript is attempted until no next right subscript exists. Any index which has been used previously is never reused. This process is illustrated in the following assuming the initial index 2.

index	$\tau(s_i, s_j)IS_{s_j}$
2	$D_2^0C_8^0A_6^0B_1^0B_4^0A_3^0$
8	$C_8^0A_6^0B_1^0B_4^0A_3^0$
6	$A_6^0B_1^0B_4^0A_3^0$
4	$B_4^0A_3^0D_2^0D_7^1$
7	$D_7^1C_8^0A_6^0B_1^0B_4^0A_3^0$
2, 8, 6, 4, 7	$D_2^0C_8^0A_6^0B_1^0B_4^0A_3^0D_2^0D_7^1C_8^0A_6^0B_1^0B_4^0A_3^0$

Each experiment generated through overlapping is now a new sub-experiment. Suppose the overlapping and merging of the previous sub-experiment resulted in the following list of sub-experiments.

2, 8, 6, 4, 7	$D_2^0C_8^0A_6^0B_1^0B_4^0A_3^0D_2^0D_7^1C_8^0A_6^0B_1^0B_4^0A_3^0$
5, 3	$D_5^1C_8^0A_6^0B_1^0B_4^0A_3^0B_1^0B_4^0A_3^0$
1	$B_1^0A_3^0D_2^0D_7^1$

Since each new sub-experiment indicates which state the machine will be in after its completion, any further concatenation of them requires that the next sub-experiment begins from that state for the sake of continuity. In this example, sequence "2, 8, 6, 4, 7" ends in state A and since sequence "1" begins with a transition from state A, the two sequences can be concatenated. In the event that a sequence does not begin with a transition from the final state of the previous sequence, a transfer sequence must be found such that the two sequences are properly connected. Hence, the transfer sequence matrix becomes useful. To concatenate "1" with "5, 3", the transfer sequence $T(D, C)$ is required.

	A	B	C	D
A	$B_1^0A_3^0$	B_1^0	$D_2^0C_8^0$	D_2^0
B	A_3^0	B_4^0	$A_3^0D_2^0C_8^0$	$A_3^0D_2^0$
C	A_6^0	$A_6^0B_1^0$	$D_5^1C_8^0$	D_5^1
D	$C_8^0A_6^0$	$C_8^0A_6^0B_1^0$	C_8^0	D_7^1

M_5 Transfer Sequence Matrix

Thus, C_8^0 is the correct transfer sequence which will perform the concatenation and would appear as "1" C_8^0 "5, 3". The final sequence is

$$D_2^0C_8^0A_6^0B_1^0B_4^0A_3^0D_2^0D_7^1C_8^0A_6^0B_1^0B_4^0A_3^0B_1^0A_3^0D_2^0D_7^1C_8^0D_5^1C_8^0A_6^0B_1^0B_4^0A_3^0B_1^0B_4^0A_3^0.$$

The overlapping and concatenating procedure is not fully developed in algorithmic form. The procedure is outlined to illustrate how it is done. Basically, it is a covering problem where each sub-experiment in the checking table must be covered in the CE. To find the shortest CE possible, the branching method (36) is utilized in the covering.

4.5 Fault Detection Algorithm

The capability of state identification sequences to verify the present state of a sequential machine suggests a procedure which can be used to detect faults in a sequential machine. This procedure is contingent on one condition, Boute (4).

Condition I

For all s_i, s_j and s_{it}, s_{jt} state pairs (t refers to a machine being tested for faults) which have identification sequences such that

$$\bar{\rho}(s_{it}, IS_{s_i}) = \bar{\rho}(s_i, IS_{s_i})$$

and

$$\bar{\rho}(s_{jt}, IS_{s_j}) = \bar{\rho}(s_j, IS_{s_j}),$$

$s_i \neq s_j$ implies $s_{it} \neq s_{jt}$.

The fault detection algorithm follows.

1. If the machine has a complete IS set, continue; otherwise redesign for a complete IS set.
2. Initialize the machine as described in Section 2.5.
3. Select a state identification sequence for each state of the machine such that Condition I is satisfied.
4. Construct the checking table for the machine.
5. Begin with the state identification sequence for the state to which the machine has been initialized and concatenate all sub-experiments contained in the checking table using overlapping reduction wherever possible.
6. The sequence yielded in step 5 is concatenated with the sequence or sequences of step 2 yielding the checking experiment.

Theorem 4.1:

Assuming that the sum of accessible and inaccessible states cannot increase due to faults in a sequential

machine and inaccessible states have been assigned entries per Section 3.6, a checking experiment derived for an n-state sequential machine using the fault detection algorithm will detect any fault or faults for that machine.

Proof:

Since Condition I is satisfied and each state variable (v) has r possible values, there are at least r^v states distinguished by the checking experiment. There must be exactly r^v states since the number of states cannot increase. Because each transition is followed by a sequence which verifies the state to which the transition takes the machine, any fault or faults are detected.

At this juncture, the restrictions on the state identification sequences imposed by Condition I are not known. A distinguishing sequence or distinguishing set (4) does satisfy Condition I. The selection of state identification sequences for the checking table must satisfy Condition I. The selection cannot be arbitrary as shown in an example which follows. In the following section some examples illustrate the fault detection algorithm.

4.6 Examples

Example 4.3

Consider M_3 which is specified as follows.

	a	b
A	A,y	B,z
B	C,z	A,z
C	B,z	C,y

M_3

	a	b
A	A_1^y	B_2^z
B	C_3^z	A_4^z
C	B_5^z	C_6^y

Working Form

Some of the state identification sequences for this machine are the following.

$$IS_A = A_1^y = a/y$$
$$IS_B = A_4^z A_1^y,\ C_3^z C_6^y = ba/zy,\ ab/zy$$
$$IS_C = C_6^y = b/y$$

The sequence $A_4^z A_1^y$ is used to identify state B and thus, yields the following checking table.

index	s_i, s_j	$\tau(s_i, s_j)$	IS_{s_j}
1	A,A	A_1^y	A_1^y
2	A,B	B_2^z	$A_4^z A_1^y$
3	B,C	C_3^z	C_6^y
4	B,A	A_4^z	A_1^y
5	C,B	B_5^z	$A_4^z A_1^y$
6	C,C	C_6^y	C_6^y

Assuming that M_3 has been initialized to state A, the checking experiment begins with A_1^y. Using this, the following sub-experiments are derived from the checking table.

$$(1) \qquad A_1^y A_1^y$$

$$(2, 4) \qquad B_2^z A_4^z A_1^y$$

$$(3, 6, 5) \qquad C_3^z C_6^y C_6^y B_5^z A_4^z A_1^y$$

These sub-experiments are concatenated to form the portion of the checking experiment after initializing. Thus,

$$\underbrace{(1)}\,\underbrace{(2, 4)}\,B_2^z\,\underbrace{(3, 6, 5)}$$

$$A_1^y A_1^y B_2^z A_4^z A_1^y B_2^z C_3^z C_6^y C_6^y B_5^z A_4^z A_1^y.$$

Since this machine does not possess a synchronizing sequence, an adaptive homing procedure can bring M_3 to state A.

This checking experiment detects any fault or faults which can occur in M_3. Now suppose state B is allowed to be identified by two different identification sequences. For index 2 let IS_B be $C_3^z C_6^y$ which exists. Thus, by allowing different identification sequences for the same state, the following experiment is constructed from the algorithm which is equally as effective.

$$A_1^y A_1^y B_2^z C_3^z C_6^y C_6^y B_5^z A_4^z A_1^y$$

Example 4.4

Consider the following machine, M_5.

	0	1
A	B,0	D,1
B	C,1	D,0
C	B,0	A,1
D	A,0	B,0

M_5

	0	1
A	B_1^0	D_2^1
B	C_3^1	D_4^0
C	B_5^0	A_6^1
D	A_7^0	B_8^0

Working Form

The state identification sequences selected for this machine are the following.

$$IS_A = D_2^1 B_8^0 = 11/10$$

$$IS_B = C_3^1 = 0/1$$

$$IS_C = A_6^1 D_2^1 = 11/11$$

$$IS_D = A_7^0 D_6^1 = 01/01$$

Since this machine has a synchronizing sequence (0101) which leaves the machine in state D, using the fault detection algorithm produces

$$0101+A^{0}_{7}D^{1}_{2}B^{0}_{8}C^{1}_{3}A^{1}_{6}D^{1}_{2}B^{0}_{8}D^{0}_{4}A^{0}_{7}D^{1}_{2}A^{0}_{7}D^{1}_{2}A^{0}_{7}B^{0}_{1}C^{1}_{3}B^{0}_{5}C^{1}_{3}.$$

This CE will detect all faults which can occur in M_5 with the exception of three sets of faults. The arrows indicate the possible starting state(s) and the faults are underlined.

	0	1
A	C,0	D,1
B	C,1	A,0
C	B,0	A,1
→D	A,0	B,0

	0	1
→A	A,1	D,1
B	C,1	D,0
→C	B,0	A,1
D	C,0	B,0

	0	1
A	C,0	D,1
B	C,1	D,0
→C	A,1	A,1
→D	A,0	B,0

These machines respond correctly to the CE only when initially in a specific state. Assuming equally likely probabilities for the machine initially in any of its n states, the probability of being in a correct state for the CE is k_i/n, where k_i is the number of correct states for the i^{th} faulty machine. The number of possible distinct configurations (d) for an n-state, p-input, q-output sequential machine satisfies, according to Harrison (20),

$$(nq)^{np}/n! < d < (nq)^{np}.$$

Suppose there exists m of these machines which satisfy the CE derived for a particular machine. Then assuming

equally likely probabilities for the existence of these m machines, due to faults, the probability (f) of any (n, p, q) faulty machine passing the CE is

$$f = m/d.$$

Taking into account that each faulty machine may not satisfy the CE when initially in all states,

$$f = 1/d(1/n \sum_{i=1}^{m} k_i)$$

or

$$(1/n(\sum_{i=1}^{m} k_i)(1/nq)^{np} < f < (1/n\sum_{i=1}^{m} k_i)(n!/(nq)^{np}).$$

For this example (n=4, p=q=2)

$$7.5 \times 10^{-8} < f < 1.8 \times 10^{-6}.$$

4.7 Bounds On Checking Experiment Length

To detect faults in sequential machines by experimental means, it is necessary to use the machine for a specified amount of time. It is then beneficial to estimate how much of the machine's time is necessary for fault detection. This section determines an upper bound on checking experiment length for checking experiments derived in Section 4.5.

Since the procedure requires that each transition in the flow table be verified, np transitions for an n-state, p-input machine are necessary. Each transition must be followed by a state identification sequence for the state in which the transition leaves the machine. Thus, there are npD inputs required (worst case) for the transition verification portion of the checking experiment, where

$$D = \left\lceil (1/n) \sum_{i} L(IS_{s_i}) \right\rceil ,$$

is the average length of the state identification sequences used.

Next, the initialization portion of the checking experiment is considered. This portion is represented as

$$L(\text{init.}).$$

The length of the CE is then

$$L(CE) = L(\text{init.}) + np(D+1).$$

This leaves the concatenation of sub-experiments to be considered. In a strongly connected n-state machine, no state can be more than (n-1) inputs distant from any other state. In the worst case, each sub-experiment in the checking table will terminate in a state (n-1) inputs distant from any other sub-experiment. The last

sub-experiment used need not be concatenated with any other and hence, concatenation of sub-experiments as an upper limit is

$$(np-1)(n-1) \text{ input symbols.}$$

Consequently an upper bound on the length of a checking experiment is

$$L(CE) = L(init.)+1+np(n+D)-n, \text{ input symbols.}$$

It has been empirically observed that length reduction, due to overlapping, compensates for additional concatenation inputs. Therefore, the following bound yields a practical guide for checking experiment length.

$$L(CE) = L(init.)+np(D+L)$$

4.8 Chapter Summary

This chapter has investigated two classes of machines and determined methods of applying experimental fault detecting principles.

This chapter introduces a notation for the sequential machine and demonstrates its usefulness by using it to remove redundancies in checking experiments. Redundancy elimination is accomplished by determining sub-experiment overlap which is covered in Section 4.4.

No previous work in the literature has been presented where transition verification is the sole test and state identification sequences are the means of verifying flow table transitions. In conjunction with the design criteria presented in Chapter 3, this dissertation presents the means of deriving checking experiments for all machines in which the number of state variables do not increase.

CHAPTER 5

FAULT DETECTION THROUGH ABSTRACT REPRESENTATIONS

A sequential machine flow table is, in fact, an abstract mathematical representation of physical components and their interconnections. The use of this representation enables us to mathematically analyze, in general, the behavior of machines and to define faults in the machine as alterations in the abstract model. In this chapter we consider the further abstraction of machines into networks which increase the knowlege of machine behavior by examination of the interactive behavior of simpler machines. Section 5.1 introduces the notion of serial and parallel machine decomposition and what this represents in terms of machine behavior. Sections 5.2 and 5.3 discuss the interaction between faults and decomposition in parallel and serial decomposition respectively. In Section 5.4 the benefits which are available through decomposition are presented.

5.1 Serial and Parallel Decomposition

When dealing with a machine whose flow table is defined, it is quite practical to simply determine the

checking experiment and apply it. Now consider a machine for which no flow table is defined or whose flow table would be impractical to define because of its great number of states. Certainly, a CDC6500 flow table would be impractical to define. For these machines the following material provides practical alternatives to the direct application of fault detection experiments.

In certain cases these machines can be partitioned in such a way that each partition represents a known sequential machine (submachine). Hartmanis and Stearns (22) have developed methods by which some machines can be decomposed into two or more component submachines.

Now it is necessary to consider whether or not testing the network of submachines, derived through decomposition, is the same as testing the machine which is represented by the network. Since Hartmanis and Stearns present the tools for machine decomposition, the bulk of that work is not duplicated here. However, Definitions 5.1-5.8 and Theorem 5.1 from Hartmanis and Stearns are used in this section.

Definition 5.1

If M and M' are two machines, then the triple (ι, K, λ) is said to be an assignment of M into M' iff

ι is a mapping of S into nonvoid subsets of S',

K is a mapping of I into I',

λ is a mapping from O' to O,

and these mappings satisfy the following relations:

1) $\eta'(\iota(s), K(x_j)) \leq \iota(\eta(s, x_j))$ for all s in S and all x_j in I,
2) $\lambda(\rho'(s', K(x_j))) = \rho(s, x_j)$ for all s' in $\iota(s)$ and x_j in I.

Definition 5.2

A machine M' is said to be a realization of machine M iff there is an assignment (ι, K, λ) of M into M'.

We can now make an observation about the behavior of M and M'.

Theorem 5.1:

If M' is a realization of M through the assignment (ι, K, λ), then for s' in $\iota(s)$ and $(\bar{x}_k)_j$ in the set of all finite input sequences

$$\bar{\rho}(s, (\bar{x}_k)_j) = \lambda(\bar{\rho}'(s', K((\bar{x}_k)_j))).$$

Thus, given any finite input sequence, the output of M' behaves like M under the mappings K and λ.

Definition 5.3

Machine M' is said to realize the state behavior of machine M iff M' realizes M with an assignment (ι, K, λ) such that ι maps each state of M onto a single state of M' and ι is one-to-one.

In this case the relations in Definition 5.1 reduce to the following:

1) $\eta'(\iota(s), K(x_j)) = \iota(\eta(s, x_j))$ for all s in S and all x_j in I,
2) $\lambda(\rho'(\iota(s), K(x_j))) = \rho(s, x_j)$ for all s in S and x_j in I.

Definition 5.4

The serial connection of two machines

$$M_a = (S_a, I_a, O_a, \eta_a, \rho_a) \text{ and } M_b = (S_b, I_b, O_b, \eta_b, \rho_b)$$

for which

$$O_a \subseteq I_b$$

is the machine

$$M = M_a \textcircled{D} M_b = (S_a \times S_b, I_a, O_b, \eta, \rho)$$

where

$$\eta((s, t), x_k) = (\eta_a(s, x_k), \eta_b(t, \rho_a(s, x_k)))$$

and

$$\rho((s, t), x_k) = \rho_b(t, \rho_a(s, x_k)).$$

Definition 5.5

The parallel connection of two machines

$$M_a = (S_a, I_a, O_a, \eta_a, \rho_a) \text{ and } M_b = (S_b, I_b, O_b, \eta_b, \rho_b)$$

is the machine

$$M = M_a /\!/ M_b = (S_a \times S_b, I_a \times I_b, O_a \times O_b, \eta, \rho)$$

with

$$\eta((s, t), (x_a, x_b)) = (\eta_a(s, x_a), \eta_b(t, x_b))$$

and

$$\rho((s, t), (x_a, x_b)) = (\rho_a(s, x_a), \rho_b(t, x_b)).$$

Assuming that a machine is properly decomposed as Hartmanis and Stearns prescribe, then we have the

following definitions.

Definition 5.6

The machine $M_a \textcircled{D} M_b$ ($M_a // M_b$) is a serial (parallel) decomposition of a machine M iff $M_a \textcircled{D} M_b$ ($M_a // M_b$) realizes M.

Definition 5.7

If $M_a \textcircled{D} M_b$ ($M_a // M_b$) is a state behavior realization of a machine M, then we say that $M_a \textcircled{D} M_b$ ($M_a // M_b$) is a serial (parallel) decomposition of the state behavior of M.

We say that the state behavior decomposition of M into $M_a \textcircled{D} M_b$ ($M_a // M_b$) is nontrivial iff M_a and M_b each have fewer states than M. To illustrate these notions of machine decomposition, consider M_6 and its serial decomposition which is given in Hartmanis and Stearns.

	0	1
1	5,1	3,0
2	3,0	4,0
3	1,0	5,1
4	2,0	3,0
5	1,0	4,0

M_6

After decomposition we have:

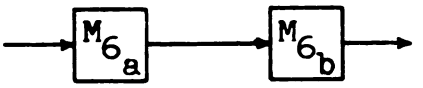
	0	1
A	B,w	B,y
B	A,y	B,w

M_{6_a}

	w	y
a	c,1	a,0
b	a,0	b,0
c	b,0	a,0

M_{6_b}

where the two machines are connected as follows.



The composite machine is M_6' and

$$M_6' = M_{6_a} \textcircled{D} M_{6_b} = (S_{6_a} \times S_{6_b}, I_{6_a}, O_{6_b}, \eta, \rho)$$

where η and ρ are given in the following table.

	0	1
(A, a)	(B, c), 1	(B, a), 0
(A, b)	(B, a), 0	(B, b), 0
(A, c)	(B, b), 0	(B, a), 0
(B, a)	(A, a), 0	(B, c), 1
(B, b)	(A, b), 0	(B, a), 0
(B, c)	(A, a), 0	(B, b), 0

M_6'

The mapping ι is:

$$1 \rightarrow (A, a),$$
$$2 \rightarrow (A, b),$$
$$3 \rightarrow (B, a),$$
$$4 \rightarrow (B, b),$$
$$5 \rightarrow (B, c).$$

The mapping κ is the same as λ which is:

$$0 \rightarrow 0,$$
$$1 \rightarrow 1.$$

We can now show that testing the network of sub-machines does test the original machine.

Theorem 5.2:

Given a serial (parallel) decomposable machine M whose state behavior is realized by $M' = M_a \textcircled{D} M_b$ ($M' = M_a /\!/ M_b$) with M_a and M_b the components of decomposition, if M_a and M_b are tested and found to be free of faults, then M is free of faults.

Proof:

Suppose M_a and M_b have been tested and are free of faults but, M is not. Then the composite machine M' ($M_a \textcircled{D} M_b$ or $M_a /\!/ M_b$) does not have a fault and M does.

We know that the composite machine M' is derived from M_a and M_b by Definition 5.4 or 5.5. Since Theorem 5.1 states that M' behaves the same as M for all finite input sequences and ι maps states of M one-to-one and onto states of M', then the determination of M_a and M_b to be free of faults also determines M to be free of faults is proven by contradiction.

The significant feature in Theorem 5.2 is that the interconnection of M_a and M_b behaves just as the original machine M does. This is true whether or not the decomposition represents internal assemblies or not. Since checking experiments are concerned with measurements on the machine's terminals, only the behavior of the machine is observed and hence, the testing of an abstract representation ($M_a \circledcirc M_b$ or $M_a // M_b$) yields the same results as testing the machine M itself.

In a fundamental theorem of machine interconnections, Hartmanis and Stearns prove that if a given machine M has certain structural properties, then there exists a network of interconnected machines, N, such that the network realizes the state behavior of M and is a natural decomposition of M. Thus, for some machines, decomposition into a network of submachines is an actual representation of the machine's construction. This also states that a network of machines used to construct a given machine M is also a decomposition of M.

Now we are able to identify a particular structure or system which previously was not evident in the machine. This increase in information concerning the machine shows us either how to design complex machines from an interconnected set of simpler machines or how to represent a given complex machine as a system of simpler machines.

The remainder of this section is used to discuss some of the structural characteristics of decomposable machines and introduce the notion of component machine testing.

Definition 5.8

A partition υ on the set of states of M is said to have the substitution property iff

$$r = t\ (\upsilon)$$

implies that

$$\eta(r, x_j) = \eta(t, x_j)\ (\upsilon)$$

for all $x_j \in I$.

Definition 5.9

If υ_1 and υ_2 are partitions on S, then $\upsilon_1 \cdot \upsilon_2$ is the partition on S such that

$$r = t\ (\upsilon_1 \cdot \upsilon_2) \text{ iff } r = t\ (\upsilon_1) \text{ and } r = t\ (\upsilon_2).$$

Definition 5.10

All finite state sequential machines are either:

a) Class I; machines whose decompositions actually represent their physical construction,
b) Class II; machines whose decompositions are not representative of their physical construction,
c) Class III; machines which cannot be decomposed.

In parallel decomposition, the next state of M_a (M_b) is independent of the conditions in M_b (M_a), i.e., the next state for M_a (M_b) depends only on the present state of M_a (M_b) and its input. Thus, the state variables of M_a and M_b are independent of each other. Hartmanis and Stearns have proven that a nontrivial parallel decomposition of a machine's state behavior exists iff two partitions (υ_1, υ_2) with substitution property such that $\upsilon_1 \cdot \upsilon_2 = \emptyset$ exist on the state set of the machine. We assume that parallel decomposable machines are of Class I as ι maps state variables of M one-to-one and onto state variables of $M' = M_a /\!/ M_b$.

In serial decomposition, Hartmanis and Stearns prove that a machine has a nontrivial decomposition of its state behavior iff a partition υ with substitution property exists on the state set of the machine. Their proof gives a method of constructing a second partition μ such that $\upsilon \cdot \mu = \emptyset$. Thus, the serial decomposition of M yields two

independent machines M_a and M_b whose state variables are independent. In each type of decomposition M_a computes the conditions for one set of state variables of M and M_b computes the conditions for the remaining set of state variables. Since M_b is constructed from μ which is not necessarily unique, then we assume that serial decompositions are generally Class II.

If M' is ($M_a // M_b$ or $M_a \textcircled{D} M_b$) and M_a and M_b are actual machines which have been interconnected to produce a larger more complex machine (design process), then we can determine that $M_a // M_b$ or $M_a \textcircled{D} M_b$ is definitely Class I and the decomposition is actually the construction.

It has been proven that testing the network of submachines will test the machine this network realizes. How does one test the submachines? This is the problem that needs answering. The answer is evident from the discussion which follows.

Machine partitioning and decomposing into a network of submachines makes available checking experiments for known portions of a machine which was initially not practical to test. However, this action does not necessarily result in trivial solutions. We must not forget that the submachines for which checking experiments are found exist within another larger machine. The total machine is the immediate environment for this machine, which could conceiveably change conditions in that environment through its responses during the experiment.

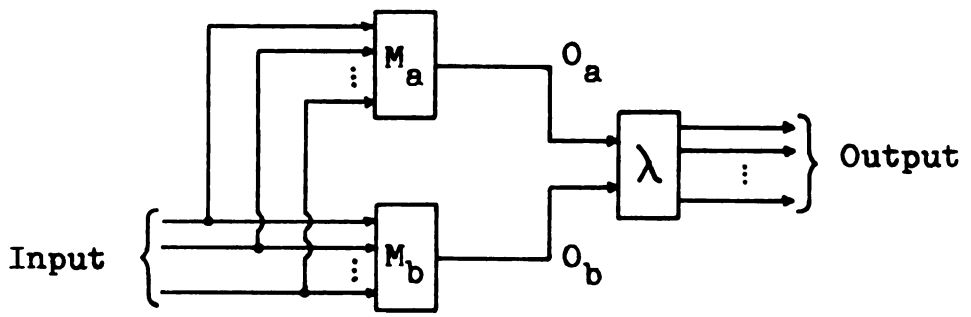
From knowledge of the overall machine, access to the submachine terminals must be ensured. Path sensitizing, Roth et al. (44), which has been developed for combinatorial circuits can be incorporated into the fault detection scheme. This would ensure that each time an input is to be applied to the submachine the configuration of the overall machine is such that the submachine is:

1) in the correct state,
2) its terminals are sensitized,
3) its outputs are observable.

If the inputs could not be sensitized and/or outputs observed, then a necessary set of testpoints and their locations could be defined as an alternative. Since parallel and serial decomposable machines are represented by different abstract models we use two different sections to investigate faults in these networks. Thus, the following two sections present the problem of faults in parallel and serial decomposable machines respectively.

5.2 Parallel Decomposition and Fault Detection

The application of a fault detection experiment to a machine which we consider to be the parallel connection of two machines (M_a and M_b) is accomplished differently than testing the machine which was originally decomposed into M_a and M_b. The structure we consider is as follows.



The box labeled λ is the combinatorial logic used to accomplish mapping $O_a \times O_b$ to the original output. Each machine is to be tested separately and thus, Theorem 5.2 ensures that the original machine is tested. Therefore, this section considers the problem of how to test each machine by examining faults and the role they play in parallel decomposition.

Das et al. (11) discuss a method for fault detection in parallel decomposable machines. Their method imposes many restrictions on the characteristics of the component machines and possible faults. Although they state that the restrictions are not stringent, the class of machines and the types of faults which can be tested greatly limit the use of their procedure or its results. The restrictions they impose are:

1) It is assumed that the component not being tested and the combinatorial circuit are free of faults,

2) It is assumed that each component machine possesses a component distinguishing sequence,
3) The machine is strongly connected,
4) No fault results in an increase in the number of states,
5) Each component machine can be placed in the desired reference state by a short adaptive experiment, a synchronizing sequence, or a special reset input.

It is true that the state variables for the component machines are independent but, when laying out the masks for integrated circuits or circuit boards one does not make a special effort to separate the state variables topographically. It is therefore not generally assumed that they cannot short together. Also, since the state variables usually act as inputs for the output logic, we would not want to assume that a fault in the output logic could not also be a fault in a component machine.

Since distinguishing sequences are not generally possessed by all sequential machines, the probability of a component not possessing such a sequence exists and thus, this procedure would be useless. The limitations imposed by restriction 4 suffers from the usual critical limitations of faults which do not increase the number of states.

In restriction 5 the emphasis is placed on short sequences. The upper bound on a synchronizing sequence

for an n-state machine, if one does exist, is $(n-1)^2 n/2$ or on the order of n^3. Since the sequence of concern here is to be applied many times in their procedure, this is very limiting. One is therefore required to rely on adaptive experiments or special reset inputs.

To overcome the very limiting restrictions of their method, the following theory is given which enables one to accomplish fault detection in both parallel and serial decomposable machines without the constraints imposed by Das et al.. Rather than attempt a general procedure, we discuss how faults in the machine map to the component machines and the inverse mapping.

Theorem 5.3:

Given that $M' = M_a /\!/ M_b$ and the only fault in the component machines is a single next state fault in M_a, then there are $|S_b|\,|I_b|$ next state faults in the table of composite machine M'.

Proof:

From Definition 5.5 we have:

$$S' = S_a \times S_b,$$
$$I' = I_a \times I_b,$$
$$\eta((r, t), (x_i, x_j)) = (\eta_a(r, x_i), \eta_b(t, x_j)).$$

Now in S' there are $|S_b|$ pairs representing states where r

is the first element and there are $|I_b|$ pairs representing inputs for M' where x_i is the first element. Thus, in $|I_b|$ columns having x_i as the first element, there is an incorrect next state entry for each of the $|S_b|$ rows having r as the first entry.

When machines M_a and M_b have their inputs physically tied together, then the (x_i, x_j) pairs, where $i \neq j$, are not allowed. Hence, the only input pairs that are allowed are (x_i, x_i). Given this arrangement, a next state fault in M_a results in $|S_b|$ faults in M' as the $|I_b|$ terms in Theorem 5.3 are dropped.

Lemma 5.1:

Given that $M' = M_a // M_b$ and the only fault in the component machines is a single output fault in M_a, then there are $S_b \; I_b$ output faults in the composite machine M'.

Proof:

This lemma is proved in a manner similar to the proof for Theorem 5.3.

Theorem 5.4:

If there is a single next state fault in machine M whose parallel decomposition ($M' = M_a // M_b$) realizes the state behavior of M, then M has a possible $|I_a| \, |S_a| + |I_b| \, |S_b| - 2$ additional next state faults.

Proof:

Given that the next state for state s_i and input x_k is a fault, then

$$\begin{aligned}\eta(s_i, x_k) &= \eta'(\iota(s_i), K(x_k)) \\ &= \eta'((r, t), (x_m, x_n)) \\ &= (\eta_a(r, x_m), \eta_b(t, x_n)).\end{aligned}$$

There are three possible cases and they are:

a) $\eta_a(r, x_m)$ is a fault,
b) $\eta_b(t, x_n)$ is a fault,
c) both a and b are true.

If a (b) is true, then there are a total of $|I_b|\,|S_b|$ ($|I_a|\,|S_a|$) next state faults in the composite machine as given in Theorem 5.3. Now if c is the case at hand, there are $|S_b|$ states in M' which have r as the first element and $|S_a|$ in which t is the second element. For the $|S_b|$ ($|S_a|$) states which have r as the first element (t as the second element) there are $|I_b|$ ($|I_a|$) inputs in M' where x_m is the first element (x_n as the second element). Now consider the state (r, t) and input (x_m, x_n). In this situation two of the next state faults occur in the same next state of M' and thus, we have $(|S_b|\,|I_b| + |S_a|\,|I_a| - 1)$ total faults in M'. Omit the original fault and there are $(|S_b|\,|I_b| + |S_a|\,|I_a| - 2)$ additional faults in M'. Since

$M_a // M_b$ is a state behavior realization of M, ι maps each state of M onto a single state of M' and ι is one-to-one. Therefore there are a possible $(|S_b|\,|I_b| + |S_a|\,|I_a| - 2)$ additional faults in M.

For machines which are actually constructed from component machines, the number of faults are those given in Theorem 5.4. The significance here is that decomposable machines exhibit patterns of faults associated with the independent state variables. This is the type of information that was previously not available in the machine M. By considering M to be realized by $M' = M_a // M_b$ we gain from structural properties otherwise hidden.

There are two ways in which we can check for faults in sequential machines. These two methods are essentially the same, only the procedure is different. These methods are:

1) Apply checking experiments to the submachines M_a and M_b to determine if they are free of faults,
2) Apply a checking experiment to the machine using knowledge of the underlying structure to derive the checking experiment.

A suggested method for fault detection in Class I machines which have a parallel decomposition is to spot check transitions in M (procedure 2). The decomposition of M

yields the necessary information on the spots (transitions) which are to be checked. The transitions which are checked are those which represent a set of transitions which would be faulty if the representative is faulty. Theorem 5.3 proves that due to the structure of the machine, there are a set of transitions which are faulty given a fault in a component machine. Thus, we perform a systematic spot checking experiment on the machine which is realized by $M_a // M_b$ using information from $M_a // M_b$.

The procedure utilizes a transition covering table which is constructed as follows.

1. Construct a k x k array (D) where k is the number of transitions available for the composite machine, i. e., $k = |S||I|$.
2. Construct the flow table for the composite machine $M_a // M_b$.
3. Label the transitions in the composite machine such that the transition for row i column j of the flow table is t_m where $m = j-1+(i-1)|I|+1$.
4. Now the entries in the array are computed using the following.

$$d_{ij} = \begin{cases} a, \text{ if } t_j \text{ is faulty when } t_i \text{ is because of } M_a, \\ b, \text{ if } t_j \text{ is faulty when } t_i \text{ is because of } M_b, \\ \text{blank, otherwise.} \end{cases}$$

We demonstrate this construction through the following example. Consider the machine M_7 and its decomposition.

	0	1
A	F,0	C,0
B	E,0	D,1
C	B,0	E,1
D	A,0	F,1
E	D,1	A,1
F	C,1	B,1

M_7

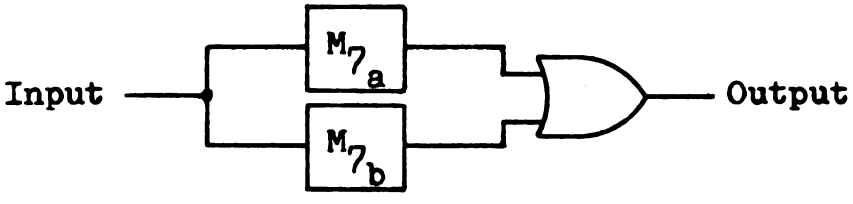
	0	1
a	c,0	b,0
b	a,0	c,1
c	b,1	a,1

M_{7_a}

	0	1
g	h,0	g,0
h	g,0	h,1

M_{7_b}

Now the composite machine M_7' is $M_{7_a} // M_{7_b}$ which is as follows.



	0	1
A → (a, g)	(c, h), 0	(b, g), 0
B → (a, h)	(c, g), 0	(b, h), 1
C → (b, g)	(a, h), 0	(c, g), 1
D → (b, h)	(a, g), 0	(c, h), 1
E → (c, g)	(b, h), 1	(a, g), 1
F → (c, h)	(b, g), 1	(a, h), 1

M_7'

	0	1
(a, g)	t_1, 0	t_2, 0
(a, h)	t_3, 0	t_4, 1
(b, g)	t_5, 0	t_6, 1
(b, h)	t_7, 0	t_8, 1
(c, g)	t_9, 1	t_{10}, 1
(c, h)	t_{11}, 1	t_{12}, 1

M_7' - Labeled Transitions

As an example of computing entries in D consider transition t_1, where

$$t_1 = \eta((a, g), 0) = (\eta_a(a, 0), \eta_b(b, 0)) = (c, h).$$

Now if t_1 is a fault because of M_{7_a}, then every state pair which has an "a" as its first element has a faulty transition when given the "0" input. Thus, in row t_1 of D an

"a" entry is made in column t_1 and t_3. If t_1 is a fault because of M_{7_b}, then every state pair which has "g" as its second entry has a faulty transition when given the "0" input. We see that in row t_1 "b" entries exist for columns t_1, t_5 and t_9. The completed array D for M_7' follows.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
t_1	ab		a		b				b			
t_2		ab		a		b				b		
t_3	a		ab				b				b	
t_4		a		ab				b				b
t_5	b				ab		a		b			
t_6		b				ab		a		b		
t_7			b		a		ab				b	
t_8				b		a		ab				b
t_9	b				b				ab		a	
t_{10}		b				b				ab		a
t_{11}			b				b		a		ab	
t_{12}				b				b		a		ab

Transition Covering Table for M_7'

If the set of transitions $\{t_2, t_3, t_5, t_8, t_{10}, t_{11}\}$ are used, then each transition in M_7' has been checked in both M_{7_a} and M_{7_b}. Thus, to check all 12 transitions for faults we need only check this set of 6 transitions.

For larger machines, the savings are more impressive as we are concerned with checking two smaller machines rather than their product.

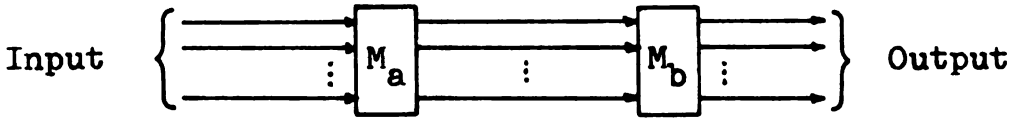
Definition 5.11

A key transition is a transition which belongs to the smallest set of transitions which cover all transitions in the transition covering table and the smallest set or sets is called a set of keys.

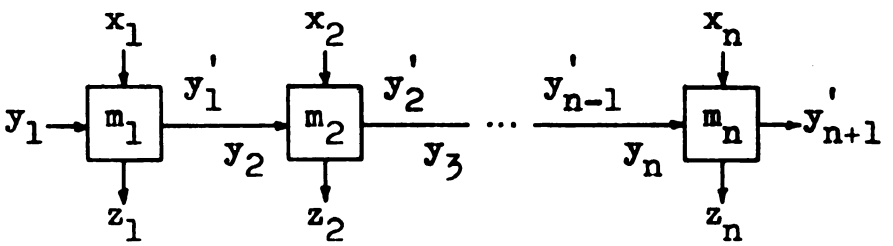
There is much more work which is suggested by the procedure which has been initiated here. The object of this section was to uncover a correspondence between parallel decomposition and faults in machines which have a parallel decomposition. This correspondence has been found and we now investigate the notion of serial decomposition and faults in machines which have a serial decomposition.

5.3 Serial Decomposition and Fault Detection

This section is concerned with faults in machines which have a serial decomposition. The structure considered in this section is the following.



The strategy is, as in parallel decomposition, to test each submachine separately to fulfill the fault detection criteria given in Theorem 5.2. A special case which fits into the realm of serial decomposition is that of iterative arrays of machines. An iterative array of machines is the interconnection of identical machines used to form a more complex structure. Breuer (7) considers a linear cascade of identical machines which appears as the following.



Breuer demonstrates how this network of identical machines when connected as a parallel adder can be checked for faults by two experiments of length 8. Thus, one does not need to check 2^{2n} additions and resets to verify that an n bit adder is free of faults.

Although this network is a serial connection of identical machines, there is the added feature that outputs (z_i) of each machine are accessible and each machine has inputs (x_i) which can be applied directly. These added features make testing for faults simpler than testing the general serial decomposition, $M_a \circledS M_b$. Kautz (30) and Menon and Friedman (38) have also investigated

faults in iterative arrays of machines.

Rather than examine serial decompositions which are of a special nature, we consider $M_a \textcircled{D} M_b$ in the general sense to illuminate possible fault detection methods which encompass a larger class of machines.

Theorem 5.5:

Given $M' = M_a \textcircled{D} M_b$ and the only fault in the component machines is a single next state fault in M_a, then there are $|S_b|$ next state faults in the composite machine M'.

Proof:

From Definition 5.4 we have:

$$S' = S_a \times S_b,$$
$$\eta((r, t), x_k) = (\eta_a(r, x_k), \eta_b(t, \rho_a(r, x_k))).$$

In S' there are $|S_b|$ pairs representing states where r is the first element. Given that $\eta_a(r, x_k)$ is the single next state fault in M_a, each of the $|S_b|$ state pairs having r as the first element would have faulty first element entries for column x_k in M'.

From Theorem 5.5 we see that a single next state fault in M_a causes M' to experience a set of faults. This is the same result which was noted in the case of parallel decomposition. The situation where next state faults

occur in M_b is different, as there are two characteristics in serial decomposition which are quite different than the parallel decomposition and they are:

1) M_b is not necessarily unique,

2) $I_b = O_a$.

Theorem 5.6:

Given $M' = M_a \textcircled{D} M_b$ and the only fault in the component machines is a single next state fault in M_b, then there are $|S_a||I|$ possible next state faults in the composite machine M'.

Proof:

Let $\eta_b(t, \rho_a(r, x_k))$ be the next state fault in M_b. Since

$$\eta((r, t), x_k) = (\eta_a(r, x_k), \eta_b(t, \rho_a(r, x_k))),$$

then every next state entry (p, q) in M' which has $q = \eta_b(t, \rho_a(r, x_k))$ is a fault. There are $|S_a|$ states in $S_a \times S_b$ where t is the second element in a state pair. The possibility exists that M_a produces the same output for all inputs x_k in I regardless of the state of M_a. Since $O_a = I_b$, every input x_k in I will transfer the $|S_a|$ states of M' having t as the second element to a state pair in which the second element is incorrect, because

$\eta_b(t, \rho_a(r, x_k))$ is incorrect.

The number $|S_a||I|$ yielded in Theorem 5.6 is derived on the basis that O_a has only one member in its alphabet. A machine which always responds with the same output regardless of the input is not very interesting and thus, it is assumed that a practical number of next state faults in M' due to a fault in M_b is $|S_a||I|-1$. We concern ourselves only with nontrivial submachines.

Because the next state function for M' is dependent on the output of M_a, the occurence of output faults in M_a can cause next state faults in M'.

Theorem 5.7:

Given $M' = M_a \textcircled{D} M_b$ and the only fault in the component machines is a single output fault in M_a, then there exist $|S_b|$ possible next state faults in the composite machine M'.

Proof:

There are $|S_b|$ states in $S_a \times S_b$ where r, a state in M_a, is the first element in a state pair. If $\rho_a^*(r, x_k)$ is the given single output fault, each of the $|S_b|$ states in M' having r as the first element of a state pair will have a next state fault in column x_k whenever

$$\eta_b(t, \rho_a(r, x_k)) \neq \eta_b(t, \rho_a^*(r, x_k)).$$

In M_b it is possible that

$$\eta_b(t, \rho_a(r, x_k)) \neq \eta_b(t, \rho_a^*(r, x_k))$$

for all $|S_b|$ states in M_b. Hence, it is possible to have $|S_b|$ next state faults in M' when a single output fault exists in M_a.

Theorem 5.8:

If all transitions in $M' = M_a \oplus M_b$ are tested for faults due to next state faults in M_a, then it is not necessary to test for faulty transitions in M' due to output faults in M_a.

Proof:

Consider the next state function in M' which is

$$\eta((r, t), x_k) = (\eta_a(r, x_k), \eta_b(t, \rho_a(r, x_k))).$$

Now if $\eta_a(r, x_k)$ is faulty we know from Theorem 5.5 that the $|S_b|$ state pairs in M' which have r as the first element will have next state faults for input x_k. Faults due to $\rho_a(r, x_k)$ can cause transitions in M' to be faults only when M' is in a state pair which has r as the first element and x_k is the input. Hence, the faulty transitions in M' which are caused by $\eta_a(r, x_k)$ being faulty would include the transitions which would be faulty due to a fault

in $\rho_a(r, x_k)$. The only difference might be the faulty next state.

Thus, if all transitions in M' are tested for faults due to next state faults in M_a, we need not check for next state faults in the transitions of M' due to output faults in M_a.

The significance in what has been given here is that in $M' = M_a \textcircled{D} M_b$ which realizes the state behavior of machine M, the underlying structure of M results in a given fault in M being a member of a set of faults. Hence, when testing one transition we actually check portions of other transitions and it is possible to find a set of keys which allow systematic spot checking in machines having a serial decomposition.

Theorem 5.9:

If there is a single next state fault in machine M whose serial decomposition ($M' = M_a \textcircled{D} M_b$) realizes the state behavior of M, then M has $|S_a|\,|I| + |S_b| - 1$ possible next state faults in addition.

Proof:

Given that the next state for state s_i under input x_k is a fault, then

$$\eta(s_i, x_k) = \eta'(\iota(s_i), \kappa(x_k))$$
$$= \eta'((r, t), x_j)$$
$$= (\eta_a(r, x_j), \eta_b(t, \rho_a(r, x_j)))$$

results in seven possible situations which are:

a) $\eta_a(r, x_j)$ is a fault,
b) $\eta_b(t, \rho_a(r, x_j))$ is a fault,
c) $\rho_a(r, x_j)$ is a fault,
d) a and b are true,
e) a and c are true,
f) b and c are true,
g) a, b and c are true.

From Theorem 5.8 it is evident that c and e can be omitted as the number of faults due to a will cover c and e. It can be concluded that the number of faults is greatest when d is the situation as d covers a, b and g. There are $|S_b|$ state pairs in M' which have r as the first element and $|S_a|$ state pairs where t is the second element. When $\eta_a(r, x_j)$ is a fault, Theorem 5.5 yields the result that $|S_b|$ faults are possible and when $\eta_b(t, \rho_a(r, x_j))$ is a fault, Theorem 5.6 yields the result that $|S_a||I|$ next state faults are possible in M'. Since one state in M' has both r and t as first and second elements respectively, there are $|S_b| + |S_a||I| - 1$ possible next state faults in M'. Since $M_a \oplus M_b$ is a state behavior realization of M, the

assignment (ι, K, λ) maps each state of M one-to-one and onto states of M'. Therefore, there are a possible $|S_b| + |S_a|\,|I| - 1$ next state faults in M.

The results obtained here are essentially the same as those obtained for parallel decomposition with the exception that the number of faults is different. We can therefore use the same methods as used in parallel decomposition. To illustrate this, consider M_6 and its serial decomposition. From M_6 we compute the following transition covering table.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
t_1	ab		a		a			b				
t_2		ab		a		a	b					
t_3	a		ab		a					b		
t_4		a		ab		a			b			
t_5	a		a		ab							b
t_6		a		a		ab					b	
t_7		b					ab		a		a	
t_8	b							ab		a		a
t_9				b			a		ab		a	
t_{10}			b					a		ab		a
t_{11}						b	a		a		ab	
t_{12}					b			a		a		ab

Transition Covering Table for M_6'

Investigation of the covering table yields the following set of keys.

$$\{t_1, t_2, t_9, t_{10}, t_{11}, t_{12}\}$$

Now that it has been shown that the structure of a machine holds information which allows it to be tested by systematic spot checking or by testing the component machines, we investigate and discuss how this is beneficial.

5.4 The Benefits of Machine Decomposition

Machine decomposition and partitioning provides benefits which make fault detection more feasible for some machines. These benefits are given in the following.

1) Fault detection experiments can be derived for machines whose flow table definition is not well defined. This is accomplished when the machine can be partitioned into a network of known submachines or is constructed with known submachines. Suppose three machines are connected in parallel and each machine has 16 states, then the composite machine has 4096 states. Although we could define a flow table for the composite, it would be impractical since we know that the composite can be checked for faults by checking each submachine.

2) Since the number of states in the composite machine is the product of the number of states of each component machine in machine decomposition, fault detection experiments are derived for smaller machines.
3) Some machines which do not possess a complete state identification sequence set can be decomposed into a system of machines where each submachine does have a complete state identification sequence set. We shall show in a later example a machine of this type.
4) Decomposition into a network of submachines may identify some submachines which are not necessarily important to test. Thus, in the event that it had been decided to test some of the submachines a savings in resourses is the result.

To illustrate the use of machine decomposition in fault detection experiments, consider the following Moore sequential machine M_8 and its decomposition which is given in Kohavi (32). Column z_i represents the output for machine i.

state	x = 0	x = 1	z_8
A	D	G	0
B	C	E	0
C	H	F	0
D	F	F	0
E	B	B	0
F	G	D	0
G	A	B	0
H	E	C	1

M_8

The parallel-serial decomposition of M_8 results in three component machines M_9, M_{10} and M_{11}.

	x = 0	x = 1	z_9
P	Q	Q	0
Q	P	P	1

M_9

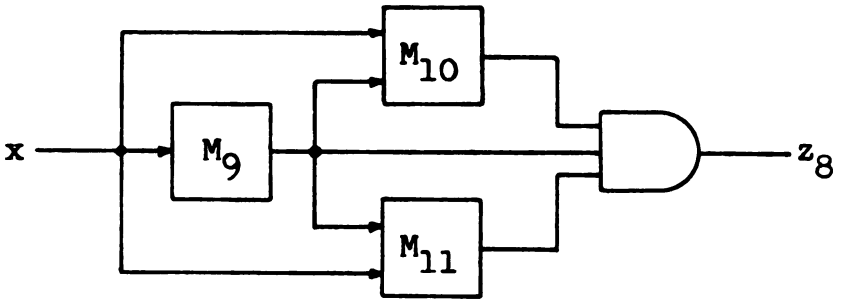
	z_9x = 00	01	10	11	z_{10}
T	T	U	U	U	0
U	U	T	T	T	1

M_{10}

	z_9x = 00	01	10	11	z_{11}
V	Y	V	V	Y	1
Y	V	Y	Y	Y	0

M_{11}

The composite structure of these three machines is as follows.



Assume that the length of a checking experiment is proportional to n^2p, where n is the number of states and p is the number of inputs. Now for the composite machine M_8 the experiment length would be 128 input symbols. For the three component machines we have for M_9, M_{10} and M_{11} that the experiment length is 8, 16 and 16 input symbols respectively.

As another illustration of benefit from decomposition, consider M_6 and its decomposition. Application of the State Identification Sequence Algorithm (Section 3.1.1) and the working form notation (Section 4.2) yields the following incomplete set of state identification sequences.

$$IS_1 = 5_1^1 = 0/1$$

$$IS_2 = 3_3^0 5_6^1 = 01/01$$

$$IS_3 = 5_6^1 = 1/1$$

$$IS_4 = 2_7^0 3_3^0 1_5^0,\ 2_7^0 3_3^0 5_6^1,\ 2_7^0 4_4^0 3_8^0$$
$$= 000/000,\ 001/001,\ 001/000$$

The two component machines M_{6_a} and M_{6_b} do have a complete set of state identification sequences and they are the following.

M_{6_a}: $IS_A = B_1^w,\ B_2^y = 0/w,\ 1/y,$

$IS_B = A_3^y,\ B_4^w = 0/y,\ 1/w.$

M_{6_b}: $IS_a = c_1^1 = w/1,$

$IS_b = a_3^0 c_1^1,\ b_4^0 a_3^0 = ww/01,\ yw/00,$

$IS_c = b_5^0 a_3^0 = ww/00.$

5.5 Chapter Summary

In this chapter it has been shown that through the use of abstract representations of a machine's structure it is possible to provide fault detection, based on that structure, which is beneficial. The benefits which can be realized have been itemized in Section 5.4. Section 5.1 views the structure theory of sequential machines with the theme that fault detection experiments are the goal.

Sections 5.2 and 5.3 consider the global characteristics of decomposition to ensure that a general theoretic approach is the result. The result is that procedures

have been outlined which are general and which can be applied to any machine which has a serial or parallel decomposition.

CHAPTER 6

ECONOMICS OF CHECKING EXPERIMENTS

Knowing how to construct checking experiments does not guarantee that fault detection theory can be economically applied. The following sections provide notions which are concerned with checking experiment application that the literature has not provided in the past. Sections 6.1 and 6.2 are concerned with the definition of costs associated with fault detection, the relationship between these costs and a cost model based on the relationship between the costs. Section 6.3 presents a procedure by which one may compute the cost of each application of a checking experiment.

The goal of this chapter is to provide criteria which consider the global analysis of fault detection experiments and thus, enable decisions concerning using fault detection or not. This chapter is concerned with Operations Research in the sense of the definition of Operations Research by Stoller (49). Operations Research is the use of the scientific method to provide criteria for decisions concerning man-machine systems involving repeatable operations.

The scientific methods are generally used in Operations Research to determine the probability of meeting specified time deadlines within cost limitations (O'Brien (41) and Hillier (26)). In this chapter we use lattice theory and provide a model of fault detection economics that defines a lattice. The fault detection economics model is new and the techniques, although different than the general methods of Operations Research, are of general interest.

6.1 Experiment Cost

Application of a checking experiment to a sequential machine is by no means free of cost. Indeed cost may make it not feasible to consider fault detection experiments in some instances. Thus, a critical step in practical fault detection is to consider experiment costs associated with:

1) machine design and preparation for testing,
2) test equipment design or purchase,
3) performing the experiment.

Along with these costs, it should be determined whether or not one would need or even want fault detection. Perhaps faults can be tolerated to some extent. Certainly, some entities which can be modeled as sequential machines have a well known behavior and hence, when the behavior changes it is not difficult to infer that a fault has

occured. There is also the situation where faults are correctable within the machine itself (Dauber (12) and Harrison (21)). The fault correcting capacity arises from the fact that once a fault causes the machine to enter an incorrect state, an input sequence exists which brings the machine back to the correct state. Thus, for some period of time the machine goes off track but, at a future time may get back on track. Perhaps these "memory lapses" can be overlooked. On the other hand, it may be absolutely necessary to know that the machine is functioning correctly.

Since reliability may or may not be necessary, one determines the cost of providing fault detection and then decides whether or not it would be profitable. If reliability is absolutely required or not required at all, the decision is trivial. In other cases a decision can be wisely made on the basis of the cost model which is provided in the following sections.

Environmental factors may or may not influence the reliability. When the sensitivity to fail is very great with regard to its environment, a machine will have a high mortality rate. When the sensitivity is small, normal failure rates due to degradation are the result. As a result of sensitivity to environment, the value of a checking experiment becomes greater and hence, its cost should be depreciated in a like manner.

6.1.1 Machine Design and Preparation for Testing

Given in this section are the particular factors which contribute to the total cost because of special machine design and preparations which must be accomplished. To facilitate fault detection, it is sometimes necessary to:

1) add one or more inputs,
2) add one or more outputs,
3) add testpoints (Williams et al. (51)).

Adding an input to the machine involves one terminal and hardware associated with the internal logic. Denote this cost as c_1. Additional outputs also require as a supplement, added terminals and hardware associated with internal logic. Denote the cost for additional outputs as c_2. For testpoints, their cost is c_3.

These costs will vary with the type of hardware used in the construction of sequential machines. For large scale integrated circuitry, additional terminals and their access to the internal structure will cost more than wired circuit boards. The associated circuitry for wired boards may cost more than that of an integrated circuit since the integrated circuit needs only to have its "masks" changed. The wired circuitry requires additional hardware for each board. Thus, these costs could be significant in regards to fault detection experiments.

6.1.2 Experiment Design

If a person is planning on using fault detection experiments, someone must write the necessary software to achieve the design. For small machines the cost will be less than for large machines where only components are to be tested. The latter type of machine requires more effort and software engineering resulting in an increase in time spent performing the design. Experiment design cost can be computed in the form of man-hours necessary to take a sequential machine which has been prepared for fault detection and yield the desired experiment.

Without the use of algorithmic procedures to aid in experiment design, the man-hours required are great. Usually, a designer applies an input, notes the response, applies another input and observes another response etc.. All of this is recorded and the I/O sequence is then considered a checking experiment. This method is, on the average, time consuming and not always complete. Experiment design cost is denoted c_4.

6.1.3 Performing the Experiment

Once the experiment is designed and machines are available for testing, the need for additional hardware to conduct the experiment enters the cost picture. If a computer is available, then some type of interactive device is necessary for the computer to cause inputs to the

machine under test and observe its outputs. We will call the interconnecting hardware costs c_5.

In the event that a computer is not available, then the cost for a device which can produce inputs and observe outputs is necessary and its cost is c_6. If a device to perform the checking experiment is not to be purchased but designed and built, this cost is also considered and is denoted c_7.

Now suppose a sequential machine is to be tested periodically, during its operation, then the time required for testing is important. This is because, during testing, the machine may not be used for other purposes. It may be that the machine is periodically free and available for testing in which case the cost would not be very important. The cost for "down time" is directly related to the length of the experiment. Since cost must be considered prior to actually deriving the experiment, one must estimate this cost according to a bound which is determined on the maximum length of an experiment. Thus, this "down time" cost can be computed as the amount of real time operation which is lost to running the checking experiment and is denoted c_8.

As an example of "down time" cost, suppose a sequential machine is actually controlling a large system. Then since the system function is dependent on the machine, the system must be made inert during the experiment. For a large chemical processing plant this inert time may be costly if

no means of controlling the system during tests exist or else the cost of a replacement is introduced.

6.2 Cost Model

From a practical standpoint we are concerned with the efficient use of available resourses to meet fault detection requirements. As always, there are certain trade-off situations which are available. As an example, if the decision is made not to add an input, then a checking experiment of greater length must be used. This decreases one cost but causes others to increase. To obtain a minimal cost for a checking experiment, one must consider all trade-off combinations.

The usefulness of the costs which have been defined is dependent on the ability to define how each cost relates to the others. The manner in which the defined costs interact is given in the following.

Definition 6.1

In the process of making decisions which concern the costs associated with fault detection, if all decisions necessary for the computation of c_j are all necessary or properly contained in the set of decisions needed for the computation of c_i, then $c_i \# c_j$.

Let c_0 denote the cost of machine preparation when not adding inputs, outputs or testpoints. Further, all

c_i $(0 \leq i \leq 8)$ are in units of dollars such that all $c_i = 0$.

Certainly, $c_j \# c_j$ for all j $(0 \leq j \leq 8)$. Thus, the relationship (#) is reflexive. Now consider the following cases.

(a) $c_i \# c_0$, for all i $(1 \leq i \leq 8)$.
(b) $c_j \# c_1$, for all j $(4 \leq j \leq 8)$.
(c) $c_j \# c_2$, for all j $(4 \leq j \leq 8)$.
(d) $c_j \# c_3$, for all j $(4 \leq j \leq 8)$.
(e) $c_k \# c_4$, for all k $(5 \leq k \leq 8)$.
(f) $c_5 \# c_6$.
(g) $c_5 \# c_7$.
(h) $c_8 \# c_5$.

Given in the following are the details for the relationships (a)-(h).

(h) Cost c_8 cannot be computed prior to c_5 as the amount of "down time" depends on hardware used to conduct the experiment and the interconnecting hardware affects the speed at which the testing equipment can interact with the machine being tested.
(f) and (g) The interconnecting hardware cannot be specified until the nature of the device which conducts the experiment is known.
(e) Until the experiment has been defined, the amount

of memory and computational ability of the testing equipment cannot be specified and hence, the interconnecting hardware and "down time".

(b), (c) and (d) The choice of inputs, outputs or testpoints affects the design of the experiment. If inputs are added to a machine to render it more easily testable, then this machine is different than one which had outputs added. In each situation a different machine is the result and the choice affects c_4 and all other c_i $(i \geq 4)$.

(a) We cannot compute c_1, c_2 or c_3 until it has been decided that we are not going to test the machine as it exists.

We observe that (#) is anti-symmetric, i.e.,

$$(c_i \# c_j) \text{ and } (c_j \# c_i), \text{ iff } c_i = c_j.$$

Since (#) is also transitive, we have that (#) defines a partial ordering.

The collection of defined costs $C = \left\{c_i : 0 \leq i \leq 8\right\}$ and the partial ordering (#) previously given constitutes a lattice $\langle C, \# \rangle$. To show that $\langle C, \# \rangle$ is a lattice we consider the following two definitions by Berztiss (3).

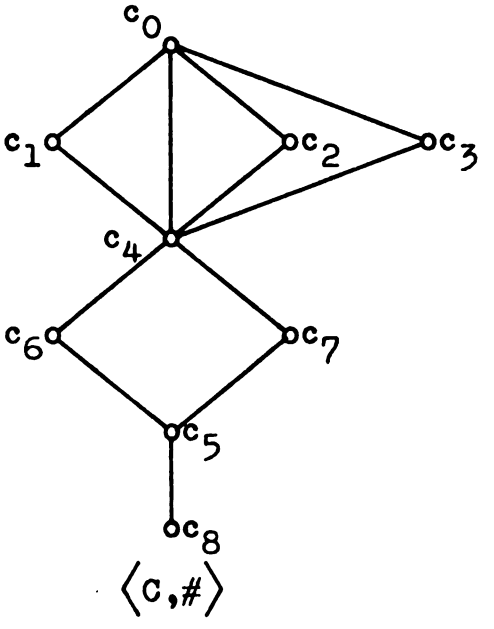
Definition 6.2

Let set A be partially ordered by # and let $\emptyset \subset B \subset A$ ($\emptyset$ is the null set). Then an element $a \in A$ is an upper bound (lower bound) of B iff, for all $b \in B$, $b \# a$ ($a \# b$). The least (greatest) element of the set of all upper (lower) bounds of B is the least upper bound (greatest lower bound) or supremum (infimum) of B, symbolized lub B (glb B) or sup B (inf B).

Definition 6.3

Let $\langle A,\# \rangle$ be a partially ordered set. The system $\langle A,\# \rangle$ is a lattice iff each pair of elements a_i and a_j of A has a supremum and an infimum in A. Denote sup $\{a_i,a_j\}$ by $a_i \bullet a_j$ and inf $\{a_i,a_j\}$ by $a_i * a_j$.

Thus, $\langle C,\# \rangle$ is a lattice as the following shows that each pair of elements in C has a supremum and an infimum.



$\langle C,\# \rangle$

This lattice clearly shows how the costs are related. Using this representation, $\langle C,\# \rangle$, we are able to construct a cost model such that one can determine the cost of applying fault detection experiments. To proceed in constructing a cost model we extract from $\langle C,\# \rangle$ rules to ensure that the cost model is representative. The following definitions are fundamental in the extraction of the rules we seek and are given in Abbot (1).

Definition 6.4

A partially ordered set $\langle P,\# \rangle$ is called totally ordered if for all p_i, $p_j \in P$ either

$$p_i \# p_j$$

or

$$p_j \# p_i.$$

Definition 6.5

If $\langle P,\# \rangle$ is totally ordered, then P is called a chain.

The lattice $\langle C,\# \rangle$ contains eight chains which span the lattice from c_0 to c_8. Each chain contains a set of costs and these eight sets are:

1) $\{c_0, c_4, c_5, c_6, c_8\}$
2) $\{c_0, c_4, c_5, c_7, c_8\}$
3) $\{c_0, c_1, c_4, c_5, c_6, c_8\}$
4) $\{c_0, c_1, c_4, c_5, c_7, c_8\}$
5) $\{c_0, c_2, c_4, c_5, c_6, c_8\}$
6) $\{c_0, c_2, c_4, c_5, c_7, c_8\}$
7) $\{c_0, c_3, c_4, c_5, c_6, c_8\}$
8) $\{c_0, c_3, c_4, c_5, c_7, c_8\}$

Consider the pairs of chains (1, 2), (3, 4), (5, 6) and (7, 8). In each pair (j, k) we observe that j and k are the same except that j contains c_6 and not c_7 whereas k contains c_7 and not c_6. It happens that each pair represents a different possible direction that one would take under machine preparation for fault detection. Also, within each pair the difference is that of how a device to perform the checking experiment is acquired. Thus, we denote each pair to be a case and the choice within each pair to be an option. This is further defined in the following definitions.

Definition 6.6

Four cases exist under machine preparation.

Case 1 The machine requires no special preparation for fault detection, i.e., no inputs, no outputs or testpoints are added.

Case 2 The machine is to have symbols assigned to its input alphabet only.

Case 3 The machine is to have symbols assigned to its output alphabet only.

Case 4 The machine will have only testpoints added to facilitate fault detection.

In case 2 we have seen (Section 3.3) how fault detection can be facilitated through the addition of one input. Fujiwara et al. (16) demonstrates how an easily testable machine can be constructed by using two additional inputs. Case 3 is covered by the material presented in Section 3.2 and Williams et al. (51) discusses a method of adding testpoints.

To each of the cases defined above, there are two options allowed.

Definition 6.7

Option 1 The experiment will be performed by use of purchased equipment.

Option 2 The experiment will be performed by use of equipment which will be designed and built.

Given the four cases and their options, the c_i are computed in the following manner.

1. Select case k ($1 \leq k \leq 4$).

2. Compute c_{k-1}.

3. Compute c_4 based on the chosen k.

4. Select an option.

5. Compute the cost of the chosen option.

6. Compute c_5 based on the option chosen.

7. Compute c_8.

Definition 6.8

An admissable cost set is a set of c_i $(0 \leq i \leq 8)$ which have been determined by the above procedure.

There are eight possible admissable cost sets. Let each admissable cost set be represented by a vector A(k), where

$$A(k) = (a_{k1}, a_{k2}, \ldots, a_{k8})$$

and

$$k = 2(i-1) + j,$$

where i and j refer to the case and option respectively. Thus, for case 4, option 1, k = 7 and

$$A(7) = (a_{71}, a_{72}, \ldots, a_{78}).$$

The entries for the kth vector, a_{km} $(1 \leq m \leq 8)$, are

the particular costs (c_m) for the case i, option j situation. As an example, consider the situation where case 3, option 1, is to be investigated. We can make the following statement:

$$a_{51} = a_{53} = a_{57} = 0.$$

This results because no inputs or testpoints are added and no device for performing the experiment will be built resulting in

$$c_1 = c_3 = c_7 = 0.$$

Thus,

$$A(5) = (0, c_2, 0, c_4, c_5, c_6, 0, c_8).$$

Consideration of all admissable cost sets yields a matrix CM. This matrix represents the cost model and appears as follows where row k is admissable cost set A(k).

$$\begin{bmatrix} 0 & 0 & 0 & a_{14} & a_{15} & a_{16} & 0 & a_{18} \\ 0 & 0 & 0 & a_{24} & a_{25} & 0 & a_{27} & a_{28} \\ a_{31} & 0 & 0 & a_{34} & a_{35} & a_{36} & 0 & a_{38} \\ a_{41} & 0 & 0 & a_{44} & a_{45} & 0 & a_{47} & a_{48} \\ 0 & a_{52} & 0 & a_{54} & a_{55} & a_{56} & 0 & a_{58} \\ 0 & a_{62} & 0 & a_{64} & a_{65} & 0 & a_{67} & a_{68} \\ 0 & 0 & a_{73} & a_{74} & a_{75} & a_{76} & 0 & a_{78} \\ 0 & 0 & a_{83} & a_{84} & a_{85} & 0 & a_{87} & a_{88} \end{bmatrix}$$

CM

Given a column matrix [I] having eight entries and each entry equal to 1, a minimal admissable cost set can be determined. Perform the following multiplication

$$[CM]\ [I] = [COST].$$

The entry of the resultant column matrix, [COST], which is smallest represents the minimal admissable cost set and is denoted C_{min}.

Example 6.1

As an example of how this cost model is generated and used to reach a cost decision, consider the following list of given information:

a) $1200 to add an input,

b) $650 to add an output,

c) $780 to add testpoints,

d) $6450 to design testing equipment,

e) $3530 to purchase testing equipment,

f) $1670 to design the experiment with no additions,

g) $490 to design the experiment with an added input,

h) $520 to design the experiment with an added output,

i) $660 to design the experiment with testpoints,

j) $1100 for interconnecting hardware case 1, opt. 1,

k) $400 for interconnecting hardware case 1, opt. 2,

l) $1400 for interconnecting hardware case 2, opt. 1,

m) $370 for interconnecting hardware case 2, opt. 2,

n) $1210 for interconnecting hardware case 3, opt. 1,

o) $260 for interconnecting hardware case 3, opt. 2,

p) $1350 for interconnecting hardware case 4, opt. 1,

q) $600 for interconnecting hardware case 4, opt. 2,

r) $3000 "down time" case 1, options 1 and 2,

s) $1000 "down time" case 2, options 1 and 2,

t) $1250 "down time" case 3, options 1 and 2,

u) $1500 "down time" case 4, options 1 and 2.

This list yields the following CM.

$$\begin{bmatrix} 0 & 0 & 0 & 1670 & 1100 & 3530 & 0 & 3000 \\ 0 & 0 & 0 & 1670 & 400 & 0 & 6450 & 3000 \\ 1200 & 0 & 0 & 490 & 1400 & 3530 & 0 & 1000 \\ 1200 & 0 & 0 & 490 & 370 & 0 & 6450 & 1000 \\ 0 & 650 & 0 & 520 & 1210 & 3530 & 0 & 1250 \\ 0 & 650 & 0 & 520 & 260 & 0 & 6450 & 1250 \\ 0 & 0 & 780 & 660 & 1350 & 3530 & 0 & 1500 \\ 0 & 0 & 780 & 660 & 600 & 0 & 6450 & 1500 \end{bmatrix}$$

Multiplication by $[I]$ yields the following cost matrix, $[COST]$.

$$\begin{bmatrix} 9300 \\ 11520 \\ 7620 \\ 9510 \\ 7160 \\ 9130 \\ 7820 \\ 9990 \end{bmatrix}$$

From this we see that the row 5 entry is smallest and hence, case 3, option 1 will yield the smallest cost.

6.3 Multiple Use of a Checking experiment

Through consideration of all trade-off conditions we are able to minimize the cost of a checking experiment. However, the cost which has been computed may seem extremely large if one were to purchase a mini-computer to perform the experiment. The situation could possibly be that even though a checking experiment is absolutely required, the cost which has been computed is prohibitive.

To achieve a true accounting of how much a checking experiment will cost, one must consider how many times the checking experiment will be used. Thus, a certain portion of the cost is distributed to each use of the experiment.

First of all, the number of machines to be examined by the checking experiment (N, $N \geq 1$) is important as the checking experiment will be used on each machine. Next, the number of times each machine will be tested (f_i, $f_i \geq 1$), where f_i refers to the ith machine. Now we have as the total number of uses of the checking experiment (T), for N machines is

$$T = \sum_{i=1}^{N} f_i.$$

Thus, the total cost C_T per use of the checking experiment is

$$C_T = C_{min}/T,$$

where the same checking experiment is used throughout.

Example 6.2

Consider the situation in which the Quikey Keyboard Company which manufactures pocket calculator keyboards decides to enter the calculator market. It has been determined that with a minimal expansion, the company can make the entire calculator. An investigation of the market reveals that dealers will not handle pocket calculators unless each one is checked for faults after assembly and prior to shipment. It is further required that one sample in each 1000 units produced be used for reduced life testing. The reduced life test requires that each sample be tested for faults once each hour for a period of 90 days.

The members of the board had decided on producing 500,000 units and felt that the company cannot enter the market as the additional financial burden of fault detection will make the product to expensive to compete. Prior to making any decision they decide to study the economics of fault detection further.

With the material given in this chapter it is concluded that C_{min} is \$23,720. For each reduced life test sample

$$f_i = (24)(90) = 2160$$

and for each regular unit

$$f_i = 2.$$

Therefore, with N = 500,000

$$T = \sum_{i=1}^{N} f_i = 500(2160) + (N-500)(2)$$

or

$$T = 2,079,000.$$

Now the resultant cost per use of the checking experiment is

$$C_T = C_{min}/T = 23,720/2,079,000 = \$.0114.$$

The resultant cost per fault detection test is \$.0114 which will not affect the calculator price enough to make it noncompetitive. Thus, it is decided to enter the pocket calculator business.

6.4 Chapter Summary

The cost of fault detection experiments is considered in this chapter. The particular costs involved in fault detection are defined and their relationships established to construct a lattice $\langle C,\# \rangle$. This lattice represents the

logical flow of events which are actually encountered in the overall fault detection scheme. From this representation, $\langle C,\# \rangle$, a cost model is derived in which one is able to determine an admissable cost set which requires minimal cost.

The final section of this chapter considers the cost per individual use of a checking experiment. This notion is useful as it allows one to determine how to defray the total cost which in some cases would otherwise seem prohibitive.

CHAPTER 7

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

7.1 Conclusions

The problem of detecting faults in sequential machines has been considered. The task of verifying the transition function, η, presents the greatest barrier to a general solution of the problem. This barrier was found to exist because experimental fault detection cannot directly observe the transition function. Thus, one must make decisions, concerning faults, by the manner in which the machine responds to experiments.

An algorithm for finding state identification sequences has been developed. Since some machines do not possess an identification sequence for each of their states, means of augmenting these machines are necessary. The methods of accomplishing this have been presented. Thus, the means of establishing all shortest, existing identification sequences for a state and the means by which a state can be given an identification sequence have been formally developed.

The problem of faults which can cause inaccessible states to become accessible has been treated. The general

problem remains open but, the case where the sum of accessible and inaccessible states remain constant has been solved in the sense that the inaccessible state assignment algorithm will yield state identification sequences for inaccessible states.

A procedure for using state identification sequences to detect faults in a sequential machine has been suggested which yields substantial results. The special Mealy machine notation, introduced in Chapter 4, contributes greatly to this procedure through reduction of experiment length by elimination of redundancies.

In Chapter 5, the correspondence which exists between faults and the structure of decomposable machines has been presented. This correspondence is of significant importance as indicated by the benefits listed in Section 5.4. It is shown that because of the underlying structure of the decomposable machine, a "systematic spot checking" procedure can be used to verify the machine's transition function. A procedure which determines which spots to check is given in Section 5.2.

The importance of the economic considerations presented in Chapter 6 is notable. With the cost model given in this chapter, decisions can be made regarding the feasibility of fault detection. Thus, Chapter 6 makes known the alternatives available to the global fault detection scheme and how to accomplish fault detection most economically.

7.2 Suggestions For Future Work

The situation in which faults can cause the sum of accessible and inaccessible states to increase needs attention. The possibility of state variables increasing and hence, the total number of accessible and inaccessible states increasing does exist and therefore must be considered.

Although the fault detecting procedure suggested in this work represents a substantial result it is not complete. To illustrate this, M_3 had a checking experiment constructed for it which has a length of 12 input symbols and the following experiment

$$A_1^y A_1^y B_2^z C_3^z C_6^y C_6^y B_5^z A_4^z A_1^y$$

whose length is 9 is just as capable in detecting faults. It is suggested that there is a procedure which can decide which state identification sequences to select for the checking table such that the optimum in conciseness results for the checking experiment.

Another area that needs work is that of defining, exactly, the criteria which state identification sequences must satisfy to qualify them for Condition I. The example worked for M_5 shows that the state identification sequences chosen do not test the machine completely. A different choice has been found which does work but, no

rules for finding them are available as of now. It is suggested that a portion of the problem rests in the fact that this machine has an input, x_m, which has

$$\eta(s_i, x_m), \rho(s_i, x_m) = \eta(s_j, x_m), \rho(s_j, x_m),$$

where $i \neq j$. This situation yields identical sub-experiments in the checking table and thus, the possibility of not checking one of the transitions.

In Section 5.2, two ways of determining faults in the class of machines which are decomposable were listed. We pursued one of these two resulting in "systematic spot checking". This "systematic spot checking" approach has only been initiated here. There is more work which needs to be accomplished to make it completely functional.

It is suggested that the other avenue to fault detection for decomposable machines (verification of submachines) merits further exploration. Two notions to consider are:

1) make the outputs of each submachine observable for fault detection,
2) make the fault detection scheme an integral part of a design procedure used to construct large complex machines from simpler ones.

BIBLIOGRAPHY

BIBLIOGRAPHY

(1) Abbot, J. C., Sets, Lattices, and Boolean Algebras, Allyn and Bacon, Boston, 1969.

(2) Abramson, N., Information Theory and Coding, McGraw-Hill, New York, 1963.

(3) Berztiss, A. T., Data Structures Theory and Practice, Academic Press, New York and London, 1971.

(4) Boute, R. T., "Distinguishing Sets for Optimal Identification in Checking Experiments," IEEE Trans. Comput., Vol. C-23, pp. 874-877, Aug. 1974.

(5) _____, "Optimal and Near Optimal Checking Experiments for Output Faults in Sequential Machines," IEEE Trans. Comput., Vol. C-23, pp. 1207-1213, Nov. 1974.

(6) Boute, R. T.; and McCluskey, E. J., "Fault Equivalence in Sequential Machines," Proc. Symposium Computers and Automata, 1971.

(7) Breuer, M. A., "Fault Detection in a Linear Cascade of Identical Machines," Proc. Ninth Annual Symposium on Switching and Automata Theory, pp. 235-243, 1968.

(8) Carter, W. C., "Fault-Tolerant Computing: An Introduction and A Viewpoint," IEEE Trans. Comput., Vol. C-22, pp. 225-229, March 1973.

(9) Chang, H. J.; Manning, E.; and Metze,G., Fault Diagnosis of Digital Systems, New York: Wiley-Interscience, 1970.

(10) Chang, L., "Fault Analysis of Combinational Logic Networks," Dept. Computer Science, Michigan State University, Technicial Rpt. 74-01, July 1974.

(11) Das, P.; and Farmer, D. E., "Fault-Detection Experiments for Parallel-Decomposable Sequential Machines," IEEE Trans. Comput., Vol. C-24, pp. 1104-1109, Nov. 1975.

(12) Dauber, P. S., "An Analysis of Errors in Finite Automata," Information and Control, Vol. 8, pp. 295-303, 1965.

(13) Farmer, D. E., "Algorithm for Designing Fault Detection Experiments," IEEE Trans. Comput., Vol. C-22, pp. 159-167, Feb. 1973.

(14) Friedman, A. D.; and Menon, P. R., "Restricted Checking Sequences for Sequential Machines," IEEE Trans. Comput., Vol. C-22, PP. 397-399, April 1973.

(15) _____, Fault Detection in Digital Circuits, Englewood Cliffs, N. J.: Prentice-Hall, 1971.

(16) Fuijwara, H.; Nagao, Y.; Sasao, T.; and Kinoshita, K., "Easily Testable Sequential Machines with Extra Inputs," IEEE Trans. Comput., Vol. C-24, pp. 821-827, August 1975.

(17) Gill, A., Introduction to the Theory of Finite State Machines, McGraw-Hill, New York, 1962.

(18) Ginsburg, S., "On the Length of the Smallest Uniform Experiment Which Distinguishes the Terminal States of a Machine," J. Assoc. Computing Machinery, Vol. 5, pp. 266-280, July 1958.

(19) Gönenc, G., "A Method for the Design of Fault Detection Experiments," IEEE Trans. Comput., Vol. C-19, pp. 551-558, June 1970.

(20) Harrison, M.. A.,, "On Asymptotic Estimates in Switching and Automata Theory," J. Assoc. Computing Machinery, Vol. 13, pp. 151-157, 1966.

(21) _____, "On the Error Corecting Capacity of Finite Automata," Information and Control, Vol. 8, pp. 430-450, 1965.

(22) Hartmanis, J.; and Stearns, R. E., Algebraic Structure Theory of Sequential Machines, Englewood Cliffs, N. J.: Prentice Hall, 1966.

(23) Hennie, F. C., "Fault Detecting Experiments for Sequential Circuits," Proc. Fith Annual Symposium on Switching Circuit Theory and Logical Design, pp. 95-110, 1964.

(24) _____, Finite State Models for Logical Machines, New York: Wiley, 1968.

(25) Hibbard, T. N., "Least Upper Bounds on Minimal Terminal State Experiments for Two Classes of Sequential Machines," J. Assoc. Computing Machinery, Vol. 8, pp. 601-612, Oct. 1961.

(26) Hillier, F. S.; and Lieberman, G. J., Introduction to Operations Research, Holden-Day, 1974.

(27) Hsieh, E. P., "Checking Experiments for Sequential Machines," IEEE Trans. Comput., Vol. C-20, pp. 1152-1166, Oct. 1971.

(28) Kane, J. R.; and Yan, S. S., "On the Design of Easily Testable Machines," Proc. IEEE 12th Annual Symposium Switching and Automata Theory, pp. 38-42, Oct. 1971.

(29) Kamal, S., "Diagnosis of Intermittent Faults in Digital Systems," Ph.D. Thesis, Michigan State University, Jan. 1973.

(30) Kautz, W. H., "Testing for Faults in Cellular Logic Arrays," Proc. 8th Annual Symposium on Switching and Automata Theory, pp. 161-174, 1967.

(31) Kime, C. R., "An Organization for Checking Experiments on Sequential Circuits," IEEE Trans. Elec. Comput., Vol. EC-15, pp. 113-115, Feb. 1966.

(32) Kohavi, Z., Switching and Finite Automata Theory, McGraw-Hill, 1970.

(33) Kohavi, Z.; and Lavallee, P., "Design of Sequential Machines with Fault Detection Capabilities," IEEE Trans. Comput., Vol. EC-16, pp. 473-484, Aug. 1967.

(34) Kohavi, I.; and Kohavi, Z., "Variable Distinguishing Sequences and Their Application to the Design of Fault-Detection Experiments," IEEE Trans. Comput. Vol. C-17, pp. 792-795, Aug. 1968.

(35) Martin, R. L., "The Design of Diagnosable Sequential Machines," Proc. Hawaii Internat. Conf. Sys. Sci., 1968.

(36) McCluskey, E. J., "Minimization of Boolean Functions," Bell Systems Tech. J., Vol. 35, pp. 1417-1444, Nov. 1956.

(37) Mealy, G. H., "A Method for Synthesizing Sequential Machines," Bell Systems Tech. J., Vol. 34, pp. 1045-1080, Sept. 1955.

(38) Menon, P. R.; and Friedman, A. D., "Fault Detection In Iterative Logic Arrays," IEEE Trans. Comput., Vol. C-20, pp. 524-535, May 1971.

(39) Moore, E. F., "Gedanken-Experiments on Sequential Machines," Automata Studies, Princeton University Press, Princeton, New Jersy, pp. 129-153, 1956.

(40) Murakami, S.; Kinoshita, K.; and Ozaki, H., "Sequential Machines Capable of Fault Diagnosis," IEEE Trans. Comput., Vol. C-19, pp. 1079-1085, Nov. 1970.

(41) O'Brien, J. J., Scheduling Handbook, McGraw-Hill, New York, 1969.

(42) Poage, J. F.; and McCluskey, E. J., "Derivation of Optimum Test Sequences for Sequential Machines," Proc. Fifth Annual Symposium on Switching Circuit Theory and Logical Design, pp. 121-132, 1964.

(43) Roth, J. P., "Diagnosis of Automata Failures: A Calculus and A Method," IBM J. Res. Develop., Vol. 10, pp. 278-291, July 1966.

(44) Roth, J. P.; Wagner, E. G.; and Patzolu, G. R., "A Formal Theory of Cubical Complexes," IBM T. J. Watson Res. Cen., Rep. N69-33559, April 1969.

(45) Salomaa, A., Theory of Automata, Pergamon Press, 1969.

(46) Seshu, S.; and Freeman, D. N., "The Diagnosis of Asynchronous Sequential Systems," IRE Trans. Elec. Comput., Vol. EC-11, pp. 459-465, Aug. 1962.

(47) Sheppard, D. A.; and Vranesic, Z. G., "Fault Detection of Binary Sequential Machines Using R-Valued Test Machines," IEEE Trans. Comput., Vol. C-23, pp. 352-358, April 1974.

(48) Stark, P. H., Abstract Automata, North Holland Publishing Co., 1972.

(49) Stoller, D. S., Operations Research: Process and Strategy, University of California Press, Berkely, 1964.

(50) Warshall, S., "A Theorem on Boolean Matrices," J. Assoc. Computing Machinery, Vol. 9, pp. 11-12, Jan. 1962.

(51) Williams, M. J. Y.; and Angell, J. B., "Enhancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic," IEEE Trans. Comput., Vol. C-22, pp. 46-60, Jan. 1973.