AN ABSTRACT OF THE THESIS OF

JOHN DOUGLAS HOLT for the degree MASTER OF SCIENCE

(Name) (Degree)

in Computer Science presented on February 4, 1975

(Major Department) (Date)

Title: THE DESIGN OF FAULT DETECTION EXPERIMENTS FOR SEQUENTIAL MACHINES USING HOMOGENEOUS DISTINGUISHING SEQUENCES

Redacted for privacy

Abstract approved: _____

Professor Robert A. Short

The design of checking experiments for sequential machines which do not initially have a distinguishing sequence is investigated. Improvements are suggested to an existing method for augmenting the output logic so that the machine acquires homogeneous distinguishing sequences. To indicate how the procedure may be implemented on a computer, elements of graph theory are applied to the design steps. A systematic process for the construction of the checking experiment using homogeneous distinguishing sequences is described. Upper and lower bounds for each segment of the experiment are derived.

The Design of Fault Detection Experiments for Sequential Machines using Homogeneous Distinguishing Sequences

by

John Douglas Holt

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Commencement June 1975

APPROVED:

Redacted for privacy

Professor of Computer Science and
Chairman of Department of Computer Science

Redacted for privacy

Dean of Graduate School

Date thesis is presented February 4, 1975

Typed by Rosslyn E. Holt for John Douglas Holt

TABLE OF CONTENTS

THE DESIGN OF FAULT DETECTION EXPERIMENTS FOR SEQUENTIAL MACHINES USING HOMOGENEOUS DISTINGUISHING SEQUENCES.

1. INTRODUCTION

In recent years a great deal of attention has been focused on methods of diagnosing faults in combinational and sequential electronic digital circuits. Many sophisticated techniques have been discovered for the detection and location of these faults. Unfortunately these techniques have not been employed to any extent in the design of electronic hardware. One of the reasons for this is that for complex circuits the algorithms become difficult and time-consuming to implement in a systematic way. In addition, many of these techniques require substantial modification of the basic design and the additional cost involved is considered unacceptable.

With the advent of integrated circuits and modularization of components the emphasis in fault diagnosis has shifted to fault detection rather than fault location. Intergate lines are now considered inaccessible and this has led to the design of fault-detection experiments which depend only upon the values at the input and output terminals of the module. The nature of the experiments treated in this paper is to apply a sequence of inputs to the module and to examine the output response in order to determine whether the machine is working correctly. It has not been possible to design experiments which cover all possible types and numbers of faults in the circuit. It is necessary to classify the machines under consideration according to the failures that the experiment is expected to detect. Any sequential machine which has the correct output

response to the experiment has a state table isomorphic to that of the given table, or has a failure not in the given class. In order to be able to program the design of these experiments some of the matrix techniques employed in graph theory are modified and applied to steps in the procedure. At the same time it is desirable that the design be simple to implement by hand, and due attention has been paid to this aspect.

A number of cost criteria are relevant in evaluating test design procedures. The following factors have been considered in the design philosophy employed in this paper:

1. The amount of additional hardware required.
2. The length of the test sequence.
3. The amount of information which must be handled to design the test.

In general, the first two of these are inversely related and so also are the last two; hence the designer has some flexibility in the relative weighting he wishes to give to each factor. The hardware module is assumed to be a sequential switching circuit. A sequential circuit is one whose next state is a function of the current inputs and the present state. The transition checking approach used here makes no specific failure assumptions which are related to the circuit realization other than the number of states. Hence these experiments are independent of the circuit realization of the sequential function, and we need be concerned only with the logical behavior of the circuit. This behavior is usually represented by a state table displaying the inputs, state transitions and outputs.

This table is referred to as the sequential machine.

The following assumptions are in effect throughout this discussion:

1. The sequential machine is of the Mealy type in which the output response is a function of the present state and the current input. A block diagram of a Mealy sequential machine is shown in Figure 1.

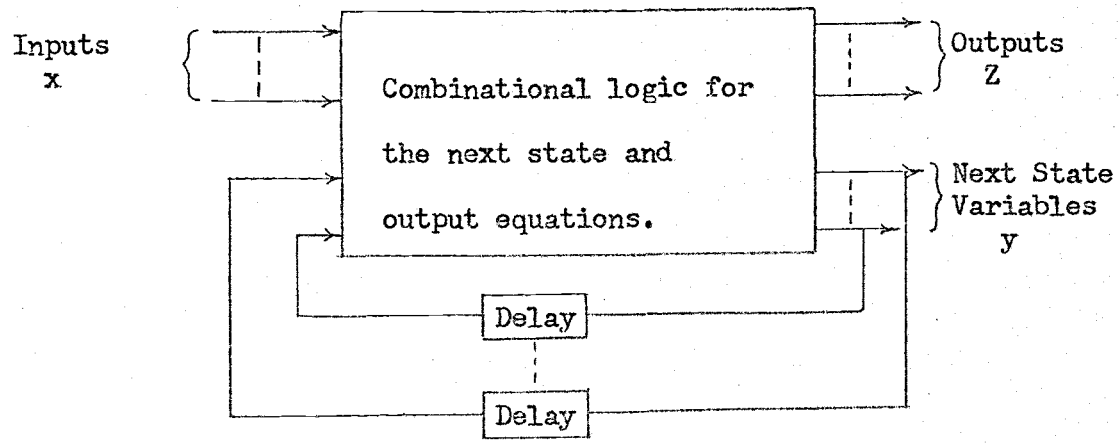


Figure 1. Mealy - Type Sequential Machine.

2. The machines are finite state, strongly connected, reduced, deterministic and completely specified and the state table is available.

3. The faults that may occur are permanent faults and ones that do not cause any increase in the number of states of the machine. This latter assumption is not too restrictive since, for the

number of states to increase, the effective number of delay elements must also increase, and this is an extremely unlikely event.

The following basic definitions apply throughout this paper.

DEFINITION: A distinguishing sequence (DS) is a sequence of inputs which, when applied to a correctly operating sequential machine (M), yields a different output response for each initial state. By observing the responses of M to the DS the initial state can be determined. A homogeneous DS is one of arbitrary length but which is constructed from only one input symbol. Not all sequential machines have a distinguishing sequence.

DEFINITION: A homing sequence is a sequence of inputs which, when applied to the inputs of M produces an output response that uniquely defines the final state, independently of the initial state. All sequential machines have a homing sequence.

DEFINITION: A synchronizing sequence is a sequence of inputs which drives M to a known final state independently of the output response. Not all sequential machines have such a sequence.

DEFINITION: A transfer sequence denoted $T(S_i,S_j)$ is a sequence of inputs which will transfer the machine from S_i to S_j. Unless otherwise specified, this is assumed to be one of the shortest such sequences. For a strongly connected machine a transfer sequence exists for every i,j.

DEFINITION: A definitely diagnosable machine is one in which every input sequence of length $n(n-1)/2$ is a DS. Often such machines will possess at least one DS that is much shorter than this.

DEFINITION: A fault detection experiment or checking experiment is a sequence of inputs which when applied to M produces an output response, examination of which will determine whether the machine is operating correctly according to its state table. The experiments are simple (only one copy of the machine is available) and preset (the entire input sequence is predetermined and cannot be influenced by the outcome of the experiment.)

Fault detection for sequential machines which do have distinguishing sequences is usually accomplished with checking experiments. The structure of such experiments is well-defined. However there does not exist an efficient algorithm for designing the sequence of input symbols. For machines which do not have distinguishing sequences, fault detection is approached from one of three possible directions:

(a) The states of the machine are partitioned in such a way that a sequence of inputs exists which distinguishes between the partition blocks. These sequences, referred to as characterizing sequences are then succeeded by a different sequence for each block which distinguishes the states in the block. The complete set of input sequences, called identifying sequences, will then identify the states of the machine. Because these identifying sequences are usually quite long, the checking experiments that they define are very long compared to the other methods available. A more recent approach Kohavi (6) has been to make use of adaptive DS's to create a set of variable length DS's which, by an appropriate choice, yield minimal checking experiments.

(b) The modification of the machine such that it will possess a minimal length homogeneous DS and will require no transfer sequences during the experiment. The modified form is referred to as an "easily testable" machine. One input is added which incorporates a permutation column. An output assignment to this column is made using Smith's Algorithm (4), thereby guaranteeing the minimal homogeneous DS involving this added input symbol. Further input columns are added until the transition graph of the machine is an Eulerian digraph, a sufficient condition for the removal of transfer sequences.

(c) The addition of outputs to the given machine until it becomes definitely diagnosable. i.e. every input sequence of length $n.(n-1)/2$ is a DS.

Only one paper has been written on the latter approach and that was the original paper by Kohavi and Lavallee (1). We found that the algorithm they presented was not systematic and did not yield good results for some sequential machines. Chapter II of this paper formalizes the procedure suggested by Kohavi and Lavallee and makes it applicable to a wider range of sequential machines. Following this we present those elements of graph theory that can greatly aid in the design and minimization of the checking experiment. We also discovered that by modifying the machine such that it acquired homogeneous DS's the problems inherent in the procedure described in Chapter II were greatly reduced in most cases and eliminated in some. In Chapter IV we investigate the modification of machines which have a column of next state entries that is either a cyclic

permutation of the original state set or is a reset column. Techniques, currently available in the literature, for achieving minimal output assignments in permutation or reset columns are extended to cover as many machines of this class as possible. The structure of the checking experiment is then described. We found that elements of a design procedure presented by Gonenc (3) were particularly suitable for the design of experiments with homogeneous DS's. The applicability of graph theory in specifying the order in which the elements of the experiment should be prescribed is emphasized. Rules for selecting input sequences of near minimal length are presented. Finally, upper and lower bounds on the length of the experiment are computed.

II. CREATING A DEFINITELY DIAGNOSABLE MACHINE

The following terms are fundamental to the discussion in this chapter:

DEFINITIONS: A successor tree for a machine and an initial set of states for this machine is a graphical display of the successor states for all possible subsets of the input alphabet applied to this initial set. A distinguishing tree (Figure 3) is a successor tree in which the states appearing at each node of the tree are grouped according to their outputs following a particular input.

DEFINITIONS: The components at each node of the tree comprise an uncertainty vector, (e.g. (115)(32) in Figure 3.) A trivial uncertainty vector is one in which each component contains only one state. A homogeneous component is a component containing repeated states, (e.g. (115) in Figure 3.)

DEFINITIONS: A testing table for machine M displays, for each possible initial pair of states, the successor state pair under all possible input/output combinations. (See Figure 4.) A testing graph is a digraph derived directly from the testing table and which has the state pairs as nodes and also shows the next state pairs when they exist. (See Figure 5.)

The procedure described below is a slight modification of Kohavi's original effort. It differs in the steps for eliminating lines from the testing graph. This method is easy to apply to machines with few states, where the cycles in the testing graph are

simple and disjoint. As the number (n) of states increases, the number of nodes in the graph grows like n(n-1) /2. For large n the solution to the problem of manipulating the lines in the testing graph must lie within the realm of graph theory and has still to be investigated.

The method given can easily be extended to allow for any number of inputs, but for simplicity it is assumed that we are dealing with a sequential machine with binary input and output values. Throughout the paper the machine shown in Figure 2 will be used as an example.

Machine M_1

Present State	Next State, Output	
	x=0	x=1
1	1,0	4,1
2	1,0	5,1
3	5,0	1,0
4	3,1	4,0
5	2,1	5,1

Figure 2. State Table for M_1.

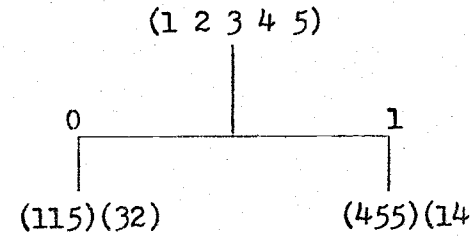


Figure 3. Distinguishing Tree For M_1.

The branches of the distinguishing tree are terminated when any of the following occur:

(1) a branch is associated with a vector containing a homogeneous component, which is a component with repeated states. e.g.

$(1,2)\rightarrow(1,1)$.

(2) an uncertainty vector in the k^{th} level appears in some branch of a preceding level. This includes self-loops. e.g. $(2,3)\rightarrow(2,3)$ as well as loops (cycles) of the type $(2,3)\rightarrow(1,5)$ $(4,5)\rightarrow(2,3)$.

If all branches of the tree terminate due to one of the above conditions then the machine has no DS.

(3) a trivial uncertainty vector occurs in which case the input sequence that leads to this branch is a DS.

Applying these tests to the distinguishing tree of M_1 we see that it has no DS because repeated states occur in response to both inputs. In order to make the machine definitely diagnosable we must prevent any branch from terminating in anything but a trivial uncertainty vector by assigning additional outputs to M_1. The testing table identifies the states that lead to homogeneous components. The testing graph highlights the loops occurring in the state pair relationships. These loops are broken by the removal of connecting lines using appropriate output assignments to the next state entries of the machine. When all homogeneous components have been separated and all loops opened, the machine will be definitely diagnosable and each branch of the distinguishing tree will eventually terminate in a trivial uncertainty vector. The length of the longest DS will be one greater than the length of the longest path in the testing graph.

For M_1, a testing table (Figure 4) is constructed by dividing the state table into its four possible combinations of input and out-

put symbols. Below this the states are taken in pairs and the balance of the table shows the implied pairs. For example, the pair (1,2) cannot be distinguished by a sequence beginning with "1" unless the pair (4,5) is distinguishable. Repeated pairs are circled. The testing graph (Figure 5) is then constructed from the uncircled pairs.

	0/0	0/1	1/0	1/1
1	1			4
2	1			5
3	5		1	
4		3	4	
5		2		5
12	(11)			45
13	15			
14				
15				45
23	15			
24				
25				(55)
34			14	
35				
45		23		

Figure 4. Testing Table for M_1.

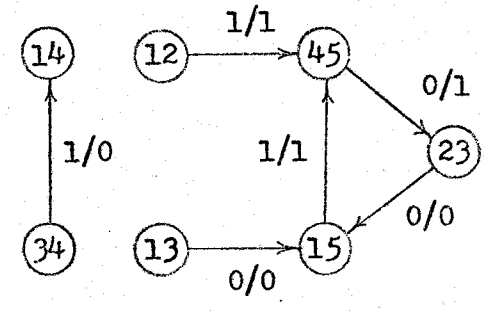


Figure 5. Testing Graph for M_1.

We now present the steps in the procedure for making M_1 definitely diagnosable. Step 1 and the first part of Step 3 comprise the original algorithm by Kohavi and Lavallee. We have added Steps 2 and 4. It is essential that Step 2 be executed before Step 3. We have also defined a set of rules that should govern the order in which the two parts of Step 3 are performed. Step 4 has been added so that outputs not specified by the first three steps may be speci-

fied in some systematic way to reduce the length of the DS.

Step 1. We must prevent the distinguishing tree from terminating due to the occurrence of homogeneous components. These components of uncertainty vectors appear as repeated entries 11 and 55 in the testing table. They must be removed by an appropriate (in this case arbitrary) assignment of different outputs to the pairs (1,2) and (2,5).

e.g.

	0	1
1	1,0<u>0</u>	4,1
2	1,0<u>1</u>	5,1<u>0</u>
3	5,0	1,0
4	3,1	4,0
5	2,1	5,1<u>1</u>

(The added output entries are shown underlined.)

N.B.1. If the given machine has two homogeneous component pairs in the same column, e.g.

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

it would be advantageous to check the two possible output assignments to determine whether one yielded further simplification for the steps to follow. There appears to be no way of avoiding an exhaustive check of the possibilities.

2. If the homogeneous component contains k identical states

then the number of additional outputs required is $\lceil \log_2 k \rceil$. The implications of more than two identical states in the homogeneous component is studied in detail in Chapter IV.

Step 2. The next step is the elimination of the condition in which a given state pair implies itself as the next state entry, e.g. (A,B)→(A,B). This situation will be displayed as a self-loop in the testing graph. The branch in the self-loop is eliminated by assigning additional outputs to the machine. This will never increase the number of output variables required by more than two, because if the two states involved in the loop have been assigned the same output value in Step 1, we can simply change the output assignment made in that step such that the homogeneous components have still been eliminated but the two states in the loop are now distinguishable. e.g. suppose that a segment of a machine after Step 1 is as follows:

1	1,0<u>1</u>
2	2,0<u>1</u>
3	1,0<u>0</u>
4	2,0<u>0</u>

The self-loop (1,2)→(1,2) can be broken without violating Step 1 by reassigning the outputs as shown:

1	1,01
2	2,0<u>0</u>
3	1,00
4	2,0<u>1</u>

There are no self-loops in M_1.

Step 3. The next step is the opening of all cycles in the testing graph by the elimination of branches using output assignments. The technique for doing this must achieve the following:

(a) the opening of all loops by the elimination of the minimal number of branches, or more specifically, by the addition of the minimal number of outputs.

(b) the elimination of branches such that the length of the longest path is minimized.

Since the length of the DS will be one greater than the length of the longest path, the goal in (b) is obvious. It will be assumed that higher priority will be given to keeping the number of additional outputs to a minimum.

For machine M_1 we have the loop $(4,5)\rightarrow(2,3)\rightarrow(1,5)\rightarrow(4,5)$. Applying step 3 (a) we see that elimination of any one of the three branches requires the minimal number of additional outputs. Hence we consider the effect of these eliminations on step 3 (b).

Eliminate	Longest path left
$(4,5)\rightarrow(2,3)$	2
$(2,3)\rightarrow(1,5)$	3
$(1,5)\rightarrow(4,5)$	3

After eliminating $(4,5)\rightarrow(2,3)$ M_1 and its testing graph have the forms shown in Figures 6 and 7.

	0	1
1	1,00	4,1
2	1,01	5,10
3	5,0	1,0
4	3,1$\underline{0}$	4,0
5	2,1$\underline{1}$	5,11

Figure 6. Modified $M_1(M_1')$.

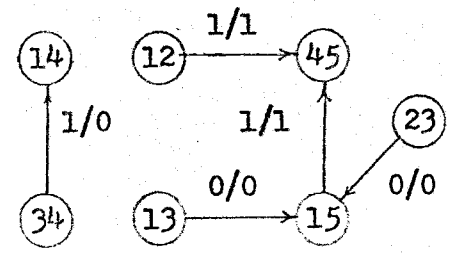


Figure 7. Testing Graph for M_1'.

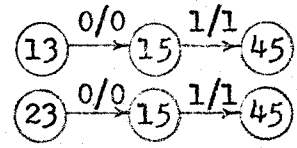
Again, it appears necessary to check all possible arrangements of the output assignments to determine whether one leads to more branch removals than another. In M_1' the choice is arbitrary.

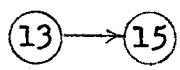
At this stage M_1' is definitely diagnosable and we could consider the unspecified entries as "don't cares." We would then have a DS of length three. However, the remaining entries can be specified only if they further reduce the length of the DS without increasing the number of outputs.

Step 4. We wish to assign remaining entries such that branches of the graph are eliminated and the remaining path lengths are reduced.

(a) Scan the first (next) input (i) column for the first (next) unspecified entry. (For M_1' this is $3 \xrightarrow{i=0} 5,0$)

(b) Examine the paths in order (longest to shortest) for a node containing this state (3) and with the outgoing branch labeled i/-.

For M_1' we have the paths 

We arbitrarily assign the output (1) to state (3) such that the 

branch is removed. (Figures 8,9.)

	0	1
1	1,0<u>0</u>	4,1
2	1,01	5,10
3	5,0<u>1</u>	1,0
4	3,10	4,0
5	2,11	5,11

Figure 8. Modified M_1' (M_1'').

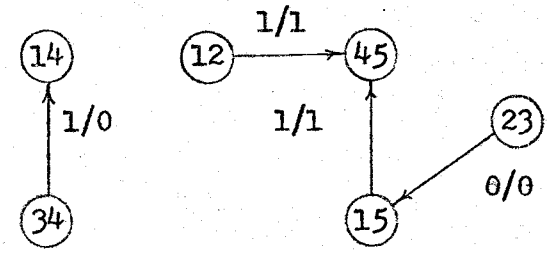


Figure 9. Testing Table for M_1''.

(c) Repeat (a) and (b) for the next unspecified state in the selected column. Continue until all columns and states have been scanned. It is possible that specification of some entries has no advantages. These entries will be left as "don't cares" in the final machine.

When step 4 has been applied to both columns in M_1' its state table, testing graph, and distinguishing tree will appear as shown in Figures 10, 11, and 12, where ϕ indicates a "don't care" specification.

	0	1
1	1,00	4,1$\underline{0}$
2	1,10	5,10
3	5,01	1,0ϕ
4	3,10	4,0ϕ
5	2,11	5,11

Figure 10. Final Form of M_1 (M_1''').

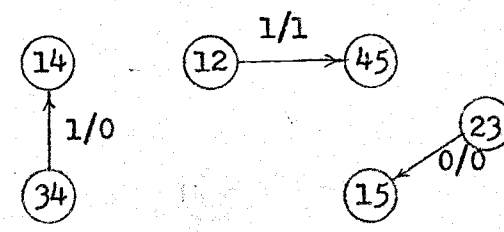


Figure 11. Testing Graph of M_1'''.

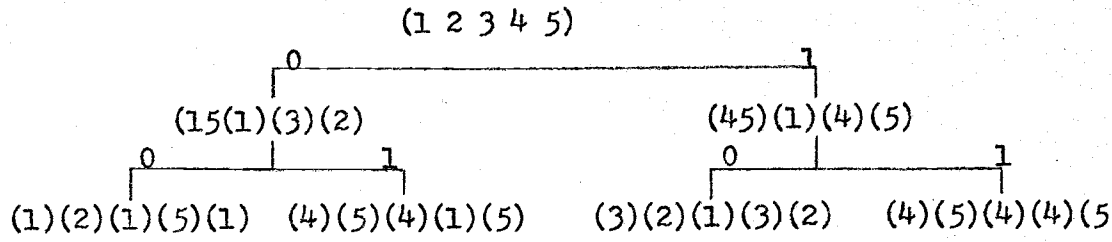


Figure 12. Distinguishing Tree for M_1'''.

N.B.1. In step 4 it is not necessary to specify an entry unless it does reduce the length of the longest path.

2. Kohavi and Lavallee (1) remark that the rules that they prescribe are only a rough guide to the order of branch cancellation and do not define it. They comment that the problem of providing an algorithm for finding the minimal number of additional required outputs is very complicated and has not been solved.

The modifications to Kohavi and Lavallee's technique that have been introduced in this paper do define the order of cancellation. However a great deal of exhaustive testing is required at each stage

in order to determine whether a particular output assignment is the best. Applying the algorithm to many "worst case" machines has so far failed to yield a non-minimal output assignment and the minimal number of additional outputs.

III. THE APPLICATION OF GRAPH THEORY

The procedure for efficiently assigning additional outputs to remove loops in the testing graph and reduce the length of the DS, described in the preceding section, is relatively easy to apply to machines with few states. If the number of states in the machine is n, the $0(n^2)$ growth in the number of nodes in the testing graph indicates the need for some programmable method for generating information about the loops (cycles) and path lengths. Some of the matrix properties and operations of directed graphs (digraphs) provide the techniques to achieve this end.

The following definitions are relevant to this section:

DEFINITION: A strong component of a digraph is one in which each node is reachable from any other node in the digraph. Thus a necessary and sufficient condition for the testing graph to have cycles is that it possess a strong component. If we can identify these strong components then we can proceed systematically to open these cycles by removing lines.

DEFINITION: A weak component of a digraph is one in which there are no cycles but there is at least one line joining each node to another in the weak component. When a line is removed to open a cycle the corresponding strong component is reduced to a weak component.

DEFINITION: A source node in a digraph is one which has no entering lines. As we will explain in Chapter VI a source node in a digraph is a desirable starting state for subsequences in the check-

ing sequence. It should be noted that although a source state does not exist for a strongly connected digraph, we will be considering digraphs displaying state transitions under subsets and extensions of the input alphabet.

The matrix techniques to be described are quite familiar aspects of graph theory (Harary (5)). However, they have not previously been applied to this particular area of switching theory. The construction of the adjacency and reachability matrices is described. The reachability matrix is then used to derive a matrix which will identify the strong components of the testing graph. When the cycles have been broken we then derive a distance matrix that will display the lengths of all paths in each weak component. A set of rules is prescribed for the selection of weak components and specific paths as targets for further analysis.

Machine M_1 is again used as an example. (Figures 13 and 14.)

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

Figure 13. M_1.

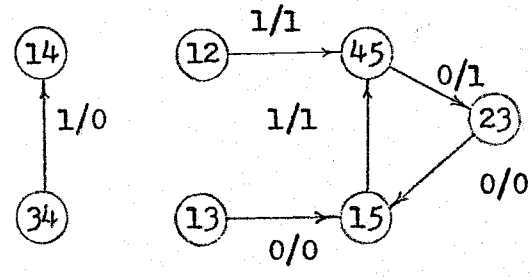


Figure 14. Testing Graph for M_1.

The Adjacency Matrix (A):

The adjacency matrix for the testing graph is constructed in the following way:

Let V_i and V_j be nodes of the digraph and $V_i V_j$ the line joining them, directed from V_i to V_j; the rows and columns of A correspond to points of the digraph (D) where $a_{ij} = 1$ if line $V_i V_j$ is in D and $a_{ij} = 0$ if $V_i V_j$ is not in D. For M_1 this leads to the adjacency matrix of Figure 15.

	12	13	14	15	23	24	25	34	35	45
12	0	0	0	0	0	0	0	0	0	1
13	0	0	0	1	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	1
23	0	0	0	1	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0
34	0	0	1	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0
45	0	0	0	0	1	0	0	0	0	0

Figure 15. Adjacency Matrix for M_1.

N.B. 1. Repeated entries in the testing table do not appear in the adjacency matrix.

2. Self-loops need not appear in A since they are always eliminated in step 2 of the procedure. Consequently, these matrix

techniques are needed for steps 3 and 4 only.

The Reachability Matrix (R):

The reachability matrix is derived from the adjacency matrix. A node V_i is reachable from V_j if there exists a path in the testing graph from V_i to V_j. The elements of $R = \{r_{ij}\}$ are defined as follows:

$r_{ij} = 1$ if V_j is reachable from V_i.

$= 1$ if $i = j$.

$= 0$ if V_j is not reachable from V_i.

N.B. A useful property of R which can be employed in the construction of checking sequences is if the i^{th} column of R is all 0's except for the diagonal element, then V_i is a source.

To obtain the reachability matrix R from the adjacency matrix A we form the following sub-sequences of matrices:

$R_0 = I$

$R_1 = (I + A)\#$ (# denotes that all operations are mod.2)

$R_2 = (I + A + A^2)\#$

$R_i = (I + A + \ldots + A^i)\#$

This sequence is generated until $R_i = R_{i+1}$, whence $R = R_i$.

For M_1 we get:

$$R_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = (I+A)\#$$

$$R_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = (R_1+A^2)\#$$

$$R_3 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = (R_2 + A^3)\# = R_4, \text{ and hence } = R$$

Figure 16. Derivation of the Reachability Matrix for M_1.

This method of computing the reachability matrix may not be the most efficient. The reader is directed to a faster algorithm described by Washall (7).

Step 3 in the method described in Chapter II involves the opening of all cycles in the testing graph. To identify these cycles (strong components) and the weak components of the testing graph that contains them we must generate the connectedness matrix. This matrix displays the connectedness category of each subset of nodes in the graph in the following way:

If entry (i,j) = 3, nodes V_i, V_j are in the same strong component.

= 2, nodes V_i and V_j are in the same weak component.

= 0, nodes V_i and V_j are not connected.

The Connectedness Matrix (C):

The connected matrix for any digraph is constructed from the reachability matrix (R) as follows:

1. Form $R + R^T$
2. Add one to each non-zero entry of $R + R^T$.

For M_1 the resulting matrix is:

	12	13	14	15	23	24	25	34	35	45
12	3	0	0	2	2	0	0	0	0	2
13	0	3	0	2	2	0	0	0	0	2
14	0	0	3	0	0	0	0	2	0	0
15	2	2	0	3	3	0	0	0	0	3
23	2	2	0	3	3	0	0	0	0	3
24	0	0	0	0	0	3	0	0	0	0
25	0	0	0	0	0	0	3	0	0	0
34	0	0	2	0	0	0	0	3	0	0
35	0	0	0	0	0	0	0	0	3	0
45	2	2	0	3	3	0	0	0	0	3

Figure 17. The Connectedness Matrix for M_1.

The strong components of M_1 can be found by examining each column of the matrix for the occurrence of two or more entries of the connectedness category "3". In Figure 17 we see that there is only one strong component containing the nodes 15, 23, 45. Columns with only one such entry may be ignored since if the corresponding

node did have a self-loop it would be removed in Step 2.

The weak components of M_1 are the intersections of all sets of nodes with the connectedness category "2" appearing in a particular column. For example, in the first column we have the set (12, 15, 23, 45), in the second (13, 15, 23, 45), and so on. The resulting intersections yield the weak components (12, 13, 15, 23, 45), (14, 34), (24), (25), (35). For the same reason stated in the previous paragraph we may ignore sets of one element.

Having now identified these components we must now decide which lines in the strong components are to be removed first. We suggest that this decision be based upon the effect of the removal of a line on the length of the longest path in the corresponding weak component. To find these lengths we need to compute the distance matrix for the weak component.

The Distance Matrix N(D):

The distance matrix of each weak component is computed from the adjacency matrix for that component.

$N(D) = \{d_{ij}\}$ where d_{ij} is the distance from node V_i to node V_j and

1. $d_{ii} = 0$,
2. $d_{ij} = \infty$ if $r_{ij} = 0$,
3. otherwise, d_{ij} is the smallest power n to which A must be raised so that $a_{ij}^{(n)} > 0$. ie. the $i,j^{\underline{th}}$ entry of $A^{n\#}$ is 1.

Procedure for Constructing N(D) from A:

1. Enter 0's on the diagonal of N(D) showing $d_{ii} = 0$.

2. Enter 1 in N(D) whenever $a_{ij} = 1$, thus showing that $d_{ij} = 1$.

3. Taking higher powers of A, whenever $a_{ij}^{(n)} = 1$ and there is no prior entry (i,j) in N(D) enter an n showing where $d_{ij} = n$.

4. Finally for some A^n every 1 will occur where there is already an entry in N(D). Enter ∞ in all remaining locations.

We will now demonstrate how the techniques described in this chapter can be used in Steps 3 and 4 of the procedure of Chapter II.

By examining the reachability matrix for M_1 we find that the strong component (15, 23, 45) consists of the lines 45→23, 23→15, and 15→45. To examine the effect of the removal of these lines on the weak component W_1 = (12, 13, 15, 23, 45) we generate the adjacency matrix for W_1. We then generate the distance matrix for each of the cases in which one of the lines of the strong component has been removed. For W_1 the results are shown in Figure 18. As can be seen, the elimination of branch 45→23 yields a weak digraph with the smallest maximum distance (2).

Branch removed: 45→23

$$\begin{bmatrix} 0 & \infty & \infty & \infty & 1 \\ \infty & 0 & 1 & \infty & 2 \\ \infty & \infty & 0 & \infty & 1 \\ \infty & \infty & \infty & 0 & 2 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

23→15

$$\begin{bmatrix} 0 & \infty & \infty & 2 & 1 \\ \infty & 0 & 1 & 3 & 2 \\ \infty & \infty & 0 & 2 & 1 \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

15→45

$$\begin{bmatrix} 0 & \infty & 3 & 2 & 1 \\ \infty & 0 & 1 & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 1 & 0 \end{bmatrix}$$

Figure 18, Distance Matrices after Branch Removal.

In general we would then select the next strong component and find the weak component in which it is contained. Branch eliminat-

ion would then be decided in the manner indicated above.

N.B. If there are several strong components in the digraph it is not clear whether the order in which they are selected by the above technique has any effect on the length of the DS obtained and the number of additional outputs required. It would seem reasonable to start with the component containing the longest path. One of the difficulties in the analysis of wuch a problem is the inability to find a sequential machine which has a testing graph of a particular form. There appears to be no systematic and efficient way of designing a digraph with the form required and then specifying the machine which has this graph.

Having opened all loops we now proceed with Step 4 - the specification of remaining entries in the state table with the aim of reducing the length of the longest path in the digraph. The current state graph and testing table for M_1 are shown in Figures 19, 20.

	0	1
1	1,00	4,1
2	1,01	5,10
3	5,0	1,0
4	3,1<u>0</u>	4,0
5	2,1<u>1</u>	5,11

Figure 19. Modified M_1.

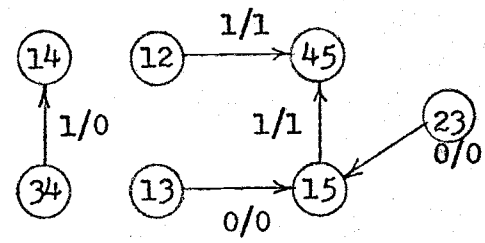


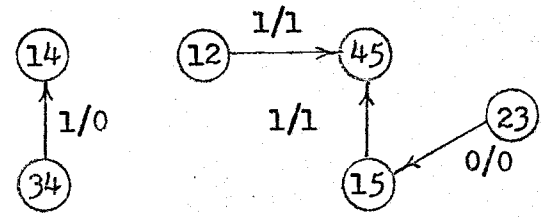
Figure 20. Testing Table for M_1.

The weak components will be the same as in the previous step because removing one line from a strong component cannot create two

disconnected weak components. The output from state 3 on an input of 0 must now be specified. From the distance graph, after the removal of the 45→23 line we see that both 13 and 23 start paths of length two on an input of 0. We arbitrarily choose 13 and eliminate the line 13→15.

At this point we must re-compute the weak components of the subset W_1 , (or in general, of the subset we are currently modifying,) because the removal of the line 13→15 may have disconnected this subset into two weak components. The state table and testing graph now have the form:

	0	1
1	1,00	4,1
2	1,01	5,10
3	5,01	1,0
4	3,10	4,0
5	2,11	5,11



All additional output entries have been assigned to the next state column for input 0. If we eliminate the line 13→15 from the adjacency matrix and re-compute the reachability and connectedness matrices for W_1 we obtain:

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} 3 & 0 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 2 & 0 \\ 0 & 0 & 2 & 3 & 2 \\ 2 & 0 & 2 & 2 & 3 \end{bmatrix}$$

From the connectedness matrix for W_1 we see that node 13 has been disconnected hence we need consider only the remaining four nodes in any further analysis. We will denote this weak component by W_1.

If we now consider the next column we find that the first unspecified output entry is in the first row. There are two nodes in W_1 containing state 1, 12 and 15. We would then generate the distance matrices for the two cases in which one of these lines is removed from W_1. These matrices would show that the removal of the line 15→45 would lead to a reduction in the maximum path length of W_1. After this is done the state table and testing graph will be as follows:

	0	1
1	1,00	4,10
2	1,01	5,10
3	5,01	1,0ϕ
4	3,10	4,0ϕ
5	2,11	5,11

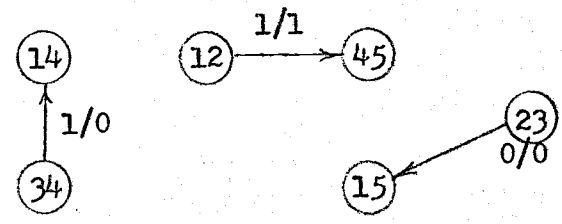


Figure 21. Final Form of the State Table and Testing Graph for M_1.

In general, we would continue this process until all entries had been specified. Each input column would be treated in turn until either all outputs have been specified or confirmed as "don't care" assignments. However, at each stage, it would be wise to ckeck whether the assignments made so far prevent the removal of any lines still remaining in W_1. In this case we would discover that the line 12→45 could not be removed without using another output variable. Therefore it is not possible to obtain a minimal path length of less than 1. We would then turn our attention to other weak components and attempt line removals only if they contained path lengths of greater than 1. For the modification of M_1 that we have in Figure 21, no more lines would be removed. Hence this is the final form of M_1 and we will have a distinguishing sequence of length two.

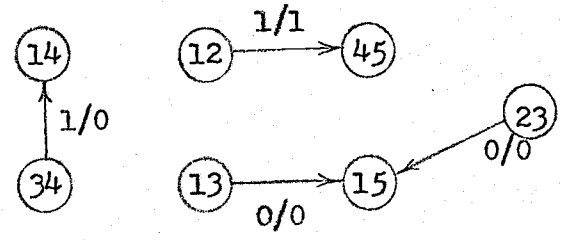
In our analysis of machine M_1 we have worked in a defined order across and down the table. We have done this only to be consistent with the method described in Chapter II. However, it should be noted that, if one is prepared to include some trial and error checking, then a better method exists for obtaining an optimal output assignment. We now outline this approach as another way of implementing Step 4.

We begin with the machine in the form shown in Figures 19 and 20 where cycles have been opened. In general, we would like to remove lines that are in the middle of paths in the weak component. We can detect the approximate position of such lines by determining the indegree and outdegree of each node in the weak component. These can be obtained from the column and row sums respectively of

the adjacency matrix. We seek a node for which the indegree is maximum and which has non-zero outdegree. If V_i is such a node then we remove the line $V_i V_j$. There will be a unique V_j if we employ the techniques to be developed in Chapter IV. It may happen however that because of previous output assignments this elimination is not possible. Then we must choose the node with next largest indegree, and so on.

If we apply this to M_1 we find that the node 15 has maximum indegree (two) and non-zero outdegree. If we remove the line 15→45 we obtain the following final form for M_1:

	0	1
1	1,00	4,10
2	1,01	5,10
3	5,0∅	1,0∅
4	3,10	4,0∅
5	2,11	5,11



This differs from Figure 21 only in that we have one more "don't care" assignment.

Clearly this method is not without its difficulties. A more detailed analysis would be required if an automated implementation of the design procedure was attempted.

In this chapter we have discussed how elements of graph theory might be employed to aid in a systematic approach to the design process. The persistent appearance of the ordering problem is one

of the major difficulties with this method. When a choice is to be made, can it be arbitrary or is there a priority of choices such that one yields a more desirable end product than another? This is not unlike the prime implicant problem or the maze problem where an exhaustive check of all possibilities appears to be the only solution. Unfortunately, exhaustive checking, particularly for machines with a large number of states, will generally be very time consuming.

Nevertheless it may be worthwhile implementing an algorithm which may not always yield minimal configurations but which is well defined in its sequence of operations and is not voracious in its time demands.

IV. HOMOGENEOUS DISTINGUISHING SEQUENCES

Considerable simplification of the methods previously described can be achieved if we do not insist that the sequential machine be definitely diagnosable but that it have DS's only for homogeneous input sequences (ie. strings of all 0's or 1's.) Homogeneous distinguishing sequences are usually most desirable for use in the design of checking experiments because they permit considerable overlap in the state counting segment of the experiment and thereby yield shorter experiments. Another significant advantage to be obtained in restricting ourselves to such DS's is that the input columns can be handled independently. There will be one testing table and testing graph for each input column. Consequently, in general, the number of nodes in a testing graph will be at least halved in the binary input case, and the possibility of having cycles in the testing graphs is greatly reduced.

Using M_1 again as the example, Steps 1 and 2 will be applied as before, after which the state table, the testing graphs for each of the two inputs, and the distinguishing tree will appear as follows:

	0	1
1	1,00	4,1
2	1,01	5,10
3	5,0	1,0
4	3,1	4,0
5	2,1	5,11

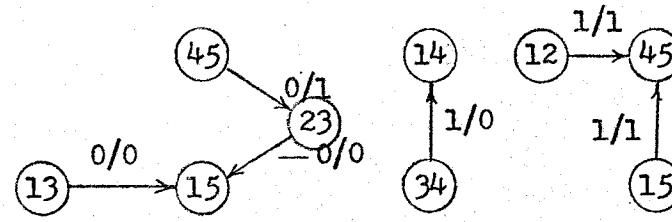


Figure 22a. State Table and Testing Graph for M_1.

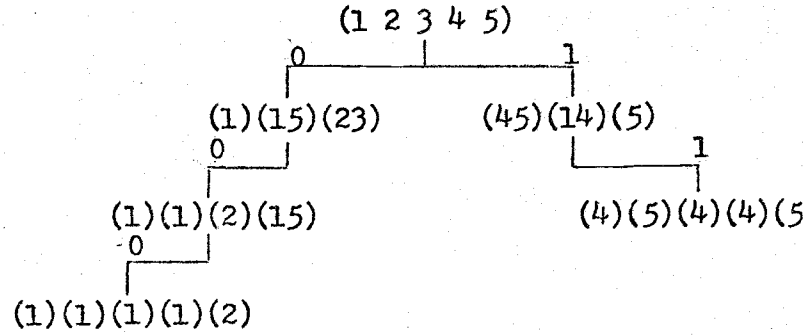


Figure 22b. Partial Distinguishing Tree for M_1.

Because each testing graph has no cycles then M_1 must have a homogeneous distinguishing sequence consisting of each of its inputs, one of length three (000) and one of length two (11). If cycles did exist in either or both of the testing graphs then we would apply Step 3 of the procedure to open them. If we had a homogeneous DS for one of the inputs only, then a decision must be made as to whether we should use the existing homogeneous DS or attempt to open the cycle in the testing graph of the other input even though an additional output variable may be necessary. As there are no cycles in M_1 we can skip Step 3 and proceed with Step 4 - the assignment of outputs to reduce the length of the longest path.

The first unassigned output is state 3 on "0". From the testing graph, and as the distance matrices sould show, the removal of the line from 23 reduces the distance to one. We accomplish this by assigning the output 0 to the entry. The next unassigned output is state 4 on "0". The removal of the branch 45→23 does not further reduce the distance hence these outputs may be left unspecified. Figure 23 displays the current form of M_1.

	0	1
1	1,00	4,1
2	1,01	5,10
3	5,00	1,0
4	3,1ϕ	4,0
5	2,1ϕ	5,11

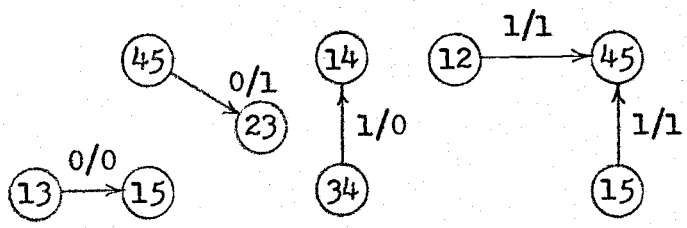


Figure 23. Modified State Table and Testing Graph for M_1.

On a "1" input the first unassigned output is state 1. A choice of 12 or 15 is arbitrary so we choose 12. Although this assignment does not reduce the maximum length it is possible that all lines will be eliminated if we proceed with the cancellation. In this case, since 25 has been specified in Step 2 and 12 specified above, then the line from 15 to 45 can never be removed. Hence the minimum distance is one and all unassigned entries, including the output for state 1, may be left as "don't cares." Final form of M_1 (M_1') is shown in Figure 24.

	0	1
1	1,00	4,1ø
2	1,01	5,10
3	5,00	1,0ø
4	3,1ø	4,0ø
5	2,1ø	5,11

Distinguishing Sequences:

00 and 11

Figure 24. Final Form of M_1.

It is clear from consideration of the testing graphs that several conclusions can be drawn about the effect of making machines diagnosable with homogeneous distinguishing sequences and that these conclusions strongly emphasize the great advantage to be obtained with this approach.

1. For any given sequential machine that there can be no DS containing mixed input symbols which is shorter than a DS containing the same symbol repeated, if the DS is obtained by the above method. This is clear from an inspection of the testing graph for the machine. Because the testing sub-graph for each input is obtained from the complete testing graph of the machine, no path in the former can be longer than a path in the latter.

2. For a sequential machine to acquire homogeneous DS's by the above method it should require no more additional outputs than to make the machine definitely diagnosable. This follows from the first conclusion. Any output assignment that opens a cycle in the complete testing graph must do the same for the testing sub-graphs.

3. In the general case, for definitely diagnosable machines, it is possible to have testing graphs in which the strong components are not disjoint. When the machine has many states this can greatly complicate the problem of eliminating a minimal number of branches to open all closed loops. However, if the machine has only two inputs, then the above method ensures that all strong components are disjoint and the elimination problem no longer exists.

4. If a machine has an input column that requires several additional output variables to make it diagnosable for that input, it may be possible to find another column which requires fewer variables. A preliminary examination of the testing subgraphs may lead to the best choice initially. It would seem reasonable to conjecture that the best choice would be an input column whose subgraph has fewest cycles, or in the event of there being an equal number of cycles the one with the shorter maximum cycle length.

5. More of the output entries in the state table are likely to remain unspecified. In most cases, when the complete testing graph is divided into the sub-graphs, the cycles will be broken and path lengths reduced. Therefore fewer output assignments will be required to obtain minimal path lengths. This will simplify the circuit realization and reduce the hardware cost.

V. SOME SPECIAL CASES

The special cases considered here are those sequential machines which have either a reset column or a cyclic permutation column. A reset column has the same state for each next state entry. A cyclic permutation column is one in which the next state sequence is some cyclic permutation of the original set of states of the machine. The reason that these machines are considered to be special is that they are the only cases for which we have some prior information about the number of additional output variables required. For reset columns this number approaches the maximum. For some of the cyclic permutations the number will be minimal, while for others the problem of assigning outputs in an optimal way has not been solved. An algorithm is presented for generating the sets of cycles in the testing graph, given the number of states and a particular cyclic permutation.

RESET COLUMNS:

Kohavi and Lavallee (1) introduced the basic technique for analysing sequential machines with reset columns. They were able to show that in certain cases, the technique of state-splitting, provided it did not increase the number of internal state variables, could make the machine definitely diagnosable. State-splitting involves the addition of a redundant state to the machine in such a way that the response of the machine to any input string is unaffected. The state added is equivalent to one of the other states in the machine and consequently the two are not distinguishable by an input sequence

of any length. An example of this technique is shown in Figures 25, 26, 27. To the original 3 state machine in Figure 25 state 3 has been added and made equivalent to state 3 by giving it the same next state and output entries (Figure 26). The original machine requires two additional outputs to make it diagnosable for homogeneous DS's. The modified version requires only one additional output (Figure 27).

	0	1
1	3,0	2,0
2	3,0	1,0
3	3,0	2,1

Figure 25. M_2.

	0	1
1	3,0	2,0
2	3,0	1,0
3	3′,0	2,1
3′	3′,0	2,1

Figure 26. State 3 is split.

	0	1
1	3,00	2,00
2	3,01	1,01
3	3′,00	2,10
3′	3′,01	2,11

Figure 27. Modified M_2.

The conditions under which state splitting is advantageous can be deduced from the above example. The number of states in the machine should be at least one less than a power of two. If m (which may be less than the number of states (n) in the machine) is the number of identical next state entries then $\lceil \log_2 m \rceil - 1$ will be the number of additional outputs required. The technique of state splitting can never save more than one additional output.

Where state splitting is not feasible then the number of additional outputs required will be $\lceil \log_2 m \rceil$. If m is close to n then in most cases the DS obtained will have length two and we will have the shortest possible experiment for this machine.

CYCLIC PERMUTATION COLUMNS:

It will be assumed throughout this section that all n states of the machine are involved in the cyclic permutation. Where the cyclic permutation involves a subset of the set of states it is suggested that the techniques to be described here should be applied to this subset and then the entire machine subjected to the procedure of Chapter IV for assigning the remaining output entries.

In this analysis of machines with columns of this type we employ the same design criteria described in Chapter I: use the smallest number of additional output variables in an assignment that minimizes the resulting homogeneous DS's. For machines with cyclic permutation columns that have the following properties:

either $f(q_j, i_r) = q_{j+1}$ and $f(q_n, i_r) = q_1$,

or $f(q_j, i_r) = q_{j-1}$ and $f(q_1, i_r) = q_n$,

where $f(q_k, i_r)$ is the next state function for state q_k on the input i_r; then it is possible to find an output assignment to the state transitions under i_r such that a minimal length homogeneous DS is obtained. Such an assignment can be made using Smith's Algorithm (4). Given the number of states (n) and the number of symbols in the output alphabet (p) this algorithm generates a sequence of output symbols such that each group of y ($= \lceil \log_2 n \rceil$) consecutive symbols, including the end-around groups, is unique. Kane and Yau (2) modified Smith's Algorithm from the original version so that it would apply directly to most machines of the type being considered here. This version is presented here for the general case where n and p are arbitrary.

SMITH'S ALGORITHM

Let n = number of states in the machine,

p = number of output symbols,

$y = \lceil \log_2 n \rceil$.

1. Order the alphabet (output) in some hierarchial manner 0,1, ..., p-1.

Begin with $0^{y-1}(p-1)$.

2. Add a new symbol if it is the highest symbol in the hierarchy which can be added without creating duplicate y-tuples in the sequence.

3. Continue step 2 until the sequence is of length n.

If $n = p^y$, STOP - this is the desired sequence.

If $n < p^y$, and if the last symbol is not equal to 0, or the last symbol equals 0 and the next to last is not equal to 0 then STOP - this is the desired sequence. Otherwise go to step 4.

4. For some value of j, $1 \leq j \leq y$, the j^{th} symbol from the end of the sequence will be other than 0 or 1. Append string $0^{y-1}1^j$ to the beginning of the sequence and delete the last (y-1)+j symbols.

N.B. The original algorithm was designed to produce zero-free sequences, ie. sequences that did not contain the string 0^y (including end around strings). The modification occurs in step 3 where we do allow the possibility of one 0^y sequence, and in step 4 where the sequence is modified if more than one 0^y string occurs.

We discovered that for certain machines with p = 2, Step 4 of Smith's Algorithm did not produce a DS at all, let alone one of minimal length. This was caused by the appearance of duplicate y-tuples. The first three such cases occurred for (y=4; n=12) and (y=5; n=20,21.) In each case it was found possible to artificially adjust the sequence so that each y-tuple was unique. It is conjectured that an output assignment using these sequences will still yield DS's with near minimal length.

We present below a full analysis of 4, 5 and 6-state machines. Smith's Algorithm is illustrated and we also highlight the problems that still exist for cyclic permutation other than those with the properties described earlier. An algorithm is given for generating all cycle sets for any n state machine with cyclic permutation columns.

FOUR STATE MACHINES:

Consider the following three possible cyclic permutation columns for four states:-

1	4,10	3,10	2,10
2	1,11	4,11	3,11
3	2,11	1,11	4,11
4	3,10	2,10	1,10
	a.	b.	c.

Figure 28a. Cyclic Permutations for 4 States.

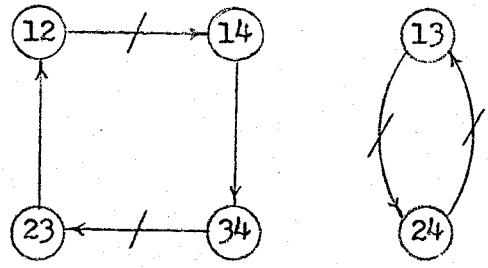


Figure 28b. Testing Graph for columns a. and c.

Figure 28b shows the testing graph for column a. The graph for column c. is essentially the same except that the arrows are reversed. Using Smith's Algorithm the output assignment will be 0110 as shown in Figure 28a. The resulting cancellations (shown / in Figure 28b) yield a DS of length two which is clearly minimal.

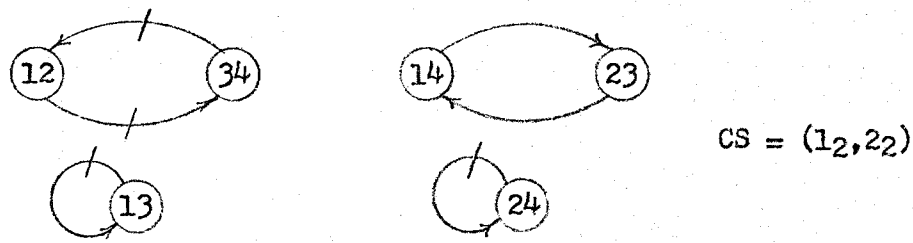


Figure 29. Testing Graph for column b.

The cycle set for column b is shown in Figure 29 together with the cancellations that would occur using the Smith Algorithm assignment. It can be shown that no arrangement of output assignments will open all closed loops. Hence an additional output variable (the maximum number in this case) is needed to enable the machine with this column to have a DS for the associated input.

FIVE-STATE MACHINES:

The cyclic permutation columns for 5 states are shown in Figure 30, and the relevant testing graphs in Figure 31.

1	5,10	4,1	3,1	2,10
2	1,10	5,1	4,1	3,10
3	2,11	1,1	5,1	4,11
4	3,11	2,1	1,1	5,11
5	4,11	3,1	2,1	1,11
	a.	b.	c.	d.

Figure 30. Cyclic Permutations for 5 states.

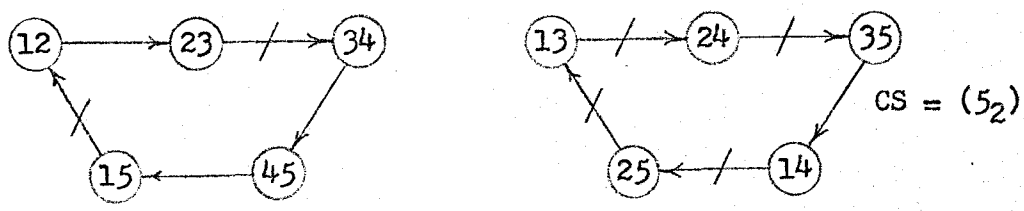


Figure 31. Testing Graphs for columns a. and d.

For columns (a) and (d) the Smith algorithm yields 00111 as an output assignment (Figure 31). The machine has a minimal homogeneous DS of length three.

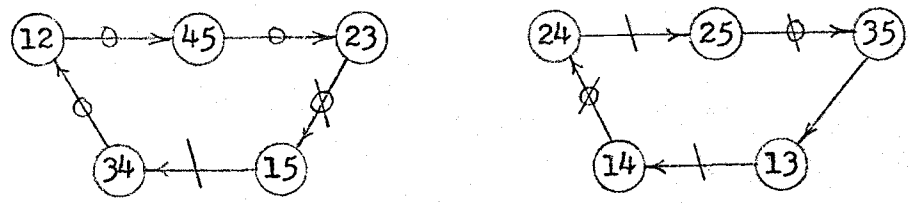


Figure 32. Testing Graphs for columns b. and c.

For columns (b) and (c) (Figure 32) the assignment 00111 produces the cancellations (shown / in Figure 32) and a DS of length four results.

This is not optimal as another output assignment was found - 01010 (shown 0) that produced a DS of length three. This latter is clearly not a sequence of the type generated by the Smith Algorithm and therefore it is highly unlikely that any modification of the Smith Algorithm could yield such a sequence. We were not able to find a systematic method for generating minimal output assignments for this case.

SIX-STATE MACHINES:

A summary of the cycle set pattern for Six-state machines is given below.

Positions that the column is rotated forward	Cycle Set
1	$(3,6_2)$
2	(3_5)
3	$(1_3,2_6)$
4	(3_5)
5	$(3,6_2)$

Output assignment using Smith's algorithm is minimal for cycle sets $(3,6_2)$ and (3_5). In fact, any assignment which contained an equal number of 0's and 1's was found to be minimal for the cycle set (3_5). No assignment yielded a DS with just one additional output for the cycle set $(1_3,2_6)$. Another output was required, but this was still less than the maximum for this number of states.

In summary, the cycle sets for 4 to 8-state machines are given in Table 1.

TABLE 1. Cycle Sets.

n:	4	5	6	7	8
Rotation					
1	$(2,4)$	(5_2)	$(3,6_2)$	(7_3)	$(4,8_3)$
2	$(1_2,2_2)$	(5_2)	(3_5)	(7_3)	$(2_2,4_6)$
3	$(2,4)$	(5_2)	$(1_3,2_6)$	(7_3)	$(4,8_3)$
4		(5_2)	(3_5)	(7_3)	$(1_4,2_{12})$
5			$(3,6_2)$	(7_3)	$(4,8_3)$
6				(7_3)	$(2_2,4_6)$
7					$(4,8_3)$

The following algorithm was formulated to generate the cycle sets for any cyclic permutation column of any sequential machine:

Let n = number of states in the machine,

x = number of positions the column has been rotated from the standard position,

then the cycle set has the general form:-

$$(a_j\ ,\ b_k)$$

where $b = \dfrac{n}{\text{gcd.}(x,n)}$

for b even:

$$a = \frac{b}{2}\ ,$$

$$j = \text{gcd}\ (x,n),$$

$$k = \frac{(n-2)}{2}.\text{gcd}\ (x,n);$$

for b odd:

$$a = b ,$$

$$j + k = \frac{(n-1)}{2}.\gcd(x,n) .$$

From the preceding analysis of these special machine classes some rather general conclusions can be drawn. Firstly, for each n state class there is at least one machine, specifically the one with each state as its own next state thereby having a testing graph with n(n-1)/2 self-loops, that will require the maximum number of additional outputs. Secondly, for each n state class, with n even, there is at least one machine that requires more than one additional output but probably less than the maximum number.

Several important problems related to these cases remain unsolved. For machines which are not covered by Smith's Algorithm there must be algorithms which will yield the most efficient output assignments. For machines with an odd number of states it appears that only one additional output is required. It is not clear whether this holds for large n. We were unable to determine whether the form of the cycle set gives any indication of the algorithm that will yield the best output assignment; or whether the form of the cycle set can be used to predict the number of additional outputs required.

In this chapter we have investigated some classes of machines about which it is possible to predict the number of additional output variables required and to make minimal output assignments. Unfortunately there exists other related classes about which we have not been

able to obtain such information. Further investigation is needed in this area, specifically with regard to the problems discussed above.

VI. DESIGN OF THE CHECKING EXPERIMENT

Having established a procedure for ensuring that a homogeneous distinguishing sequence exists for the sequential machine we now consider the design of a checking experiment employing this sequence. The checking experiment must verify the number of states of the machine and check the transitions from these states under all inputs. If at any point in the experiment the output response differs from the expected response as indicated by the state table of the machine then we can conclude that the machine is not functioning correctly. An algorithm is presented, based on work by Gonenc (3) but modified here to take into account the advantageous properties of the homogeneous DS, that will yield a near minimal length checking experiment for most sequential machines.

The method to be described here requires that the machine be in a predetermined state. To achieve this we may first apply any one of several types of sequences. A synchronizing sequence, if one exists, will drive the machine to a final state which is known from the state table but which must be tested as the first step in the experiment. Alternately, we may apply a homing sequence which will leave the machine in a known and recognized state because the corresponding output does uniquely identify the final state. If the output response was not as expected then we have determined that the machine is faulty and the experiment terminates. If we want the machine in a state that is not reachable directly by a homing sequence then we must first apply a homing sequence and

follow it with a sequence that will transfer the machine to the desired state. Again, for this latter case it will be necessary to recognize the start state at the beginning of the experiment.

It has become common practice in designing checking experiments to consider the experiment in two parts - the α-sequence in which the number of distinct states in the machine is verified, and a β-sequence in which the state transitions are checked. This partitioning will be used here since the incorporation of homogeneous distinguishing sequences in the α-sequence yields substantial reductions in the length of the experiment. It should be noted however, that if the correct response to a DS is observed during the transition checking portion of the experiment then this step may be eliminated from the state counting segment. This means that, in general, the checking experiment can be based solely on checking transitions, since for a strongly connected machine the correct number of distinct output responses will be observed, thereby verifying the number of states in the machine.

The following symbols are used in this chapter:

X_d = the distinguishing sequence.

$1 - X_d$ = an input 1(0) followed by the distinguishing sequence.

$T(S_i,S_j)$ = a shortest transfer sequence from S_i to S_j.

Z_j = the output response sequence that distinguishes state S_j.

For an example machine we will use our previous machine M_1 which was modified in chapter IV to exhibit DS's of 00 and 11.

	0	1
1	1,00	4,1ϕ
2	1,01	5,10
3	5,00	1,0ϕ
4	3,1ϕ	4,0ϕ
5	2,1ϕ	5,11

Figure 33. Modified M_1''.

Organization of the α-Sequence:

The goal of the first part of the checking experiment is to confirm that the number of states, verified by successive applications of the DS to the machine being tested, agrees with the number initiated by the state table. One application of the DS and its corresponding output response will be referred to as a cell. It follows that the number of distinct cells should correspond to the number of states in the state graph. Because we are using a homogeneous DS there will be considerable overlap of state checking cells in the α-sequence.

The first step in the design of the α-sequence is to construct a digraph (Figure 34) displaying the next state transitions for the input used in the DS. For this example we will use the DS of "00" for machine M_1''.

A desirable start state for the α-sequence is one which minimizes the number of transfer sequences required. A source state, if one exists, satisfies this requirement since if we were to choose a non-source state then at some point in the sequence we would have

to apply a transfer sequence to tha source state so that we could check its output response to the DS. If no source exists then the digraph is strongly connected and the particular start state chosen is immaterial. If more than one source state exists then we should choose the one that minimizes the total length of all transfer sequences needed in the α-sequence.

If the transition graph has disjoint segments then transfer sequences will be required. The optimal strategy would be to choose the source states in an order that minimizes the total length of all transfer sequences required. This situation would be further complicated if for one of the segments there were multiple source states. There appears to be no way of selecting á priori the best order of states.

For machine M_1'' state 4 is the only source state and is therefore the best start state. If any other state was chosen as the initial state then at some point in the α-sequence a transfer sequence to state 4 would be required. In addition, the overlapping effect of the DS would be lost.

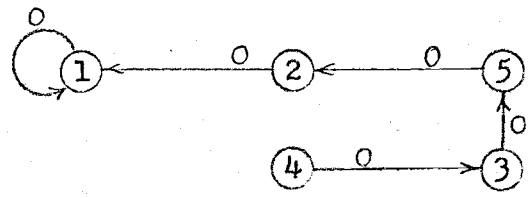


Figure 34. M_1'' State Transitions for input 0.

The α-sequence of this machine will therefore have the following form:

Input	0	0	0	0	0	0	0	
State	4	3	5	2	1	1	1	1
Output	1ϕ	00	1ϕ	01	00	00	00	

The first six inputs produce five different output responses, thereby verifying that the machine has five distinct states. The final input causes a transition that has already occurred and been tested. Consequently, provided the machine is not faulty, at the end of the α-sequence we are in a recognized final state. If the machine is faulty we may still have the same output response at this stage, but the remainder of the experiment will detect the fault.

In general, following the correct number of different output responses it will be necessary to apply the distinguishing sequence again in order to repeat a transition and terminate in a known state.

The procedure for designing the α-sequence is now summarized:

1. If the given sequential machine has more than one homogeneous distinguishing sequence, choose one of the shortest.

2. Choose one of the source states under the same input symbol used in the DS. If there is no source state, choose any state as the start. Where there is a choice of start states whether it is due to no source state or multiple source states, it would probably be better to choose as the start state one of the states that can be reached with one of the shortest transfer sequences.

3. Apply the DS.

4. If the state reached has not yet had a distinguishing sequence applied to it (ie. has not been recognized) then go to step 3. Otherwise go to step 5.

5. If there is a source state not yet recognized apply one symbol of the DS and then a transfer sequence to reach that state, then go to step 3. Otherwise go to step 6.

6. If there is any state not yet recognized apply one symbol of the DS and then a transfer sequence to reach that state, then go to step 3. Otherwise go to step 7.

7. Apply the distinguishing sequence once more, in order to reach a known final state. The structure of the α-sequence is now completed.

In the above algorithm, prior to each introduction of a transfer sequence, one input symbol of the same type as is used in the DS is applied. This is done to ensure that not only does the α-sequence check the number of states but it checks all the transitions under this same input symbol. It is this property of the homogeneous distinguishing sequence that gives it such a great advantage over all other types of distinguishing sequences used in the design of checking experiments.

Again in the case where there are several sources, the question arises: does one order in which sources are chosen yield a shorter checking experiment than another? The order chosen should be dependent on the ease with which we can reach the start state, the total length of the transfer sequences required, and

the suitability of the final state as a start state for the β-sequence. The inclusion of these considerations would be quite straightforward and systematic, requiring exhaustive testing of all the possibilities. The effort and time required to do this, in most cases, will probably outweigh the advantages of the potential reduction in the experiment length. Therefore they are not included in the algorithm given above.

Furthermore, because of the great variability in the form of sequential machines it is desirable to have a procedure which is generally applicable to all machines, and which does not attempt to cater to this variability.

Organization of the β-Sequence:

The function of the β-sequence is to check the state transitions on all remaining inputs. The general approach is to apply an input (i) to produce the transition and then to recognize the state reached by applying the DS (X_d). This combination of inputs will be referred to as a cell and its form can be represented as shown:

Input		i	X_d
State	S_i		S_j
Output		Z	Z_j

When a state is reached for which all transitions have been checked it will be necessary to apply a transfer sequence to a state which still has unchecked transitions.

The construction of the β-sequence will be illustrated for M_1'' and then the algorithm formulated for machines with several inputs.

As we have checked all the 0 transitions for M_1'' in the sequence it remains to check all of the 1 transitions. It the sequence 1 had been used as a transfer sequence in the first part of the experiment the transition it created could be ignored in this second part. A graph is constructed showing all of the next state entries on an input of 1-Xd. If a transition had already been checked then the line representing it would be removed from the graph.

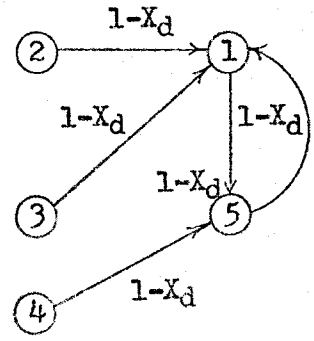


Figure 35. $1\text{-}X_d$ Graph for M_1''.

The graph in Figure 35 has three sources. In general this will indicate that three transfer sequences will be needed in the β-sequence. However, if one of the source states is a terminal state for the α-sequence we would need one fewer transfer sequence. This latter condition does not apply to M_1'', consequently three transfer sequences will be required. In this case it makes little difference whether we commence checking transitions from state 1 or first transfer to a source state.

Using state 1 as the start state, the general form of the sequence will be as follows:

Input	1 - Xd	1 - Xd	T(1,4)	1 - Xd	T(5,2)	1 - Xd
State	1	5	1	4	5	2
Input	T(1,3)	1 - Xd				
State	1	3				

The order in which the source states were chosen was based on the minimum distance between the current state and the remaining source states. Specifically, the β-sequence will be:

Checks:	1 on 1		5 on 1			4 on 1			2 on 1				3 on 1					
Input:	1	0	0	1	0	0	1	1	0	0	0	1	0	0	1	0	1	0	0
State:	1	4	3	5	5	2	1	4	4	3	5	2	5	2	1	4	3	1	1	1
Output:	1ø	1ø	00	11	1ø	01	1ø	0ø	1ø	00	1ø	10	1ø	01	1ø	1ø	0ø	00	00

The total length of the checking experiment for machine M_1'' is 26. Any experiment based on the original machine M_1 would require substantially more input symbols.

A general procedure permitting any number of additional input columns, is now presented for the organization of the β-sequence. It is a modification of an algorithm devised by Gonenc. Gonenc's algorithm has been changed to take into account the fact that we are employing homogeneous DS's. Elements of graph theory have also been introduced to indicate the techniques necessary in a programmed version of the procedure.

Let d_i^+ , d_i^- represent the outdegree and indegree respectively of node i in the graph, and S the set of states (nodes). The

nodes (S_i) of the graph will belong to one of the following sets:

$$R = \{S_i \in S \mid d_i^+ = d_i^-\}$$

$$F = \{S_i \in S \mid d_i^+ > d_i^-\}$$

$$P = \{S_i \in S \mid d_i^+ < d_i^-\}$$

We wish to select our start states for each segment of the experiment such that the use of transfer sequences is minimized. If the graph is Eulerian, that is, $d_i^+ = d_i^-$ for all nodes, then there exists a path starting at any node which travels along each line exactly once and returns to the start node. Consequently no transfer sequences will be required. If the graph is not Eulerian then at least one transfer sequence will be required. The selection of optimum sequences is based on the following well-known theorem:

THEOREM:

If digraph D is connected but not Eulerian every minimal covering of D consists of k paths each of which joins a vertex in F to one in P, where

$$k = \sum_{S_i \in F} (d_i^+ - d_i^-) = \sum_{S_i \in P} (d_i^- - d_i^+)$$

The procedure described below is for connected graphs or for connected subsets of a nonconnected graph. The minimal covering for a nonconnected graph will be the union of the minimal coverings for the connected components.

Procedure for finding a Minimal Covering:

1. Choose a start state in F. The terminal state of the pre-

vious portion of the experiment may be chosen as the start state. If it is not in F then k + 1 paths will be required and k transfer sequences used.

2. Follow a path from the start state, erasing each line as it is used. This would require elimination of the appropriate entry in the adjacency matrix. If a line chosen is a bridge (a bridge line is one whose removal will degenerate a connected graph to a non-connected graph), choose another line emanating from the same node, if one exists. Matrix techniques for determining bridges and finding alternate paths are discussed in the Appendix.

3. When it is not possible to go further, choose another start state in F and go to step 2. When k paths have been generated the procedure is terminated.

N.B. The minimal covering sequence will be one of the set of sequences generated by the above procedure. There will be at least k members of the set and it is necessary to generate all of them if the minimal covering is to be found. In general the reduction to be achieved by forming all possible coverings is probably not worth the effort. The following criteria for generating the covering is suggested as a means of achieving near-minimal results most of the time:

Compute the distance matrix for the digraph. List the maximum path lengths emanating from the nodes in F. The start states may then be chosen from F either in order of decreasing path length or such that the lengths of the transfer sequences between paths is in an increasing order. It is not clear which of these two

alternatives yield the better results. In the latter case the lengths of the transfer sequences would be obtained from the distance matrix of the state graph of the machine.

As a final example a checking experiment will be designed for the following sequential machine M_3.

	0	1
1	2,0	6,0
2	3,0	5,1
3	4,1	2,0
4	5,0	1,0
5	6,0	4,1
6	2,1	5,1

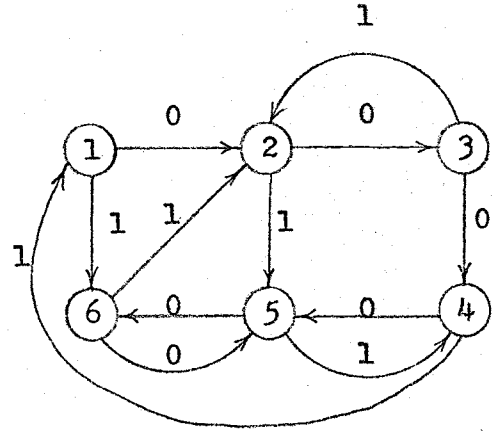


Figure 36. State Table and State Graph of M_3.

M_3 already has distinguishing sequences, one of the shortest being "001". In the search for homogeneous DS's the testing graph for each input is constructed (Figures 37 and 38).

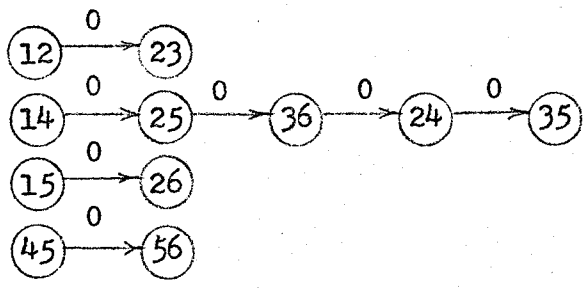


Figure 37. Input 0.

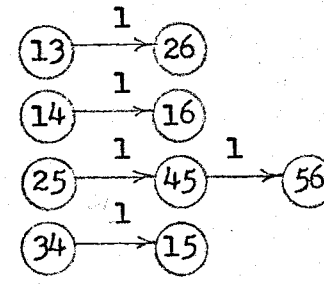


Figure 38. Input 1.

An homogeneous DS of "00000" exists. The 1 input column has the repeated pair '55' which, if removed, will give the machine a distinguishing sequence of "11". Consequently with the addition of only two output entries M_3 adquires a short DS. The remaining entries may be left as "don't cares" giving maximum flexibility in design. The modified machine M_3 is shown in Figure 39.

	0	1
1	2,0ϕ	6,0ϕ
2	3,0ϕ	5,10
3	4,1ϕ	2,0ϕ
4	5,1ϕ	1,0ϕ
5	6,0ϕ	4,1ϕ
6	2,1ϕ	5,11

Figure 39. Modified M_3 (M_3').

Construction of the α-sequence is guided by the source states in a graph of state transition on a 1 input. From Figure 36 it can be seen that 3 is the only source state. Hence, if it is chosen as the start state, no transfer sequences will be required and the sequence of states will be 3 2 5 4 1 6 5 4 1. The complete α-sequence is shown below:

Checks:		3 on 1 / 2 on 1	5 on 1 / 4 on 1	1 on 1 / 6 on 1				
Input:	1	1	1	1	1	1	1	1
State:	3	2	5	4	1	6	5	4	1

The last two inputs are necessary to drive the machine to a known state.

For the β-sequence we require the transitions on an input of 0 followed by the distinguishing sequence (Xd). The "0-Xd" graph is given in Figure 40.

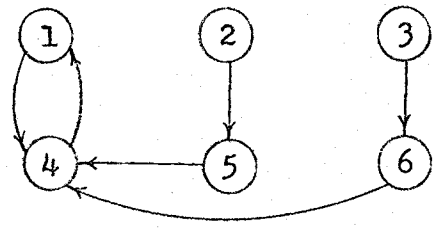


Figure 40. 0-X_d Graph.

In this case it makes no difference in the length of the β-sequence whether we choose the terminal state of the α-sequence as the start state for this segment or one of the source nodes of the 0-Xd graph. Choosing 1 as the start state the minimal covering will be 1 4 1 - 2 5 4 - 3 6 4. Two transfer sequences are required and there will be six 0-Xd cells checking the 0 transitions of all states. The complete β-sequence is now given:

Checks:	1 on 0			4 on 0				2 on 0			5 on 0							3 on 0		
Input:	0	1	1	0	1	1	0	0	1	1	0	1	1	1	0	0	0	1	1	
State:	1			4			1	2			5			4			3			6

Checks:	6 on 0			
Input:	0	1	1	
State:	6			4

A minimal checking experiment for M_3 has been designed with a length of 30 input symbols. For the same original machine with no modifications Gonenc obtained a minimal experiment of length 52 using the shortest distinguishing sequence.

VII. BOUNDS ON THE LENGTH OF THE EXPERIMENT

(a) Upper Bounds:

In deriving the upper bounds for this type of checking experiment use is made of the assumption that the machine is strongly connected and that consequently any state can be reached from any other state in at most n-1 state transitions. The bounds are computed for a two input machine, however an extension to more than two inputs would be quite straightforward.

Let L = the length of the DS.

n = number of states in the machine.

For the α-sequence the worst case occurs when there is no overlap between input cells (a cell is L + 1 inputs of the same symbol as the DS.) This cell must be applied n times. A further application of L - 1 symbols that comprise the DS is needed to drive the machine to a known terminal state. In addition, n-1 transfer sequences will be needed. Because the machine is strongly-connected the first transfer sequence will require at most one symbol, the second at most two symbols, and so on.

$\therefore$ Maximum length of the α-sequence

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

$$= n(L+1) + (L-1) + \frac{(n-1)(n-2)}{2}$$

For the β-sequence a cell consisting of one input symbol followed by the DS must be applied to each of the n states. The total length of the transfer sequences will again be $\sum_{i=1}^{n-1} i$.

$\therefore$ maximum length of the β-sequence

$$= n.(L+1) + \sum_{i=1}^{n-1} i$$

$$= n.(L+1) + \frac{(n-1).(n-2)}{2}$$

$\therefore$ upper bound on experiment length

$$= 2n(L+1) + (L-1) + (n-1).(n-2)$$

If we permit only the minimum number of additional outputs to make the machine diagnosable then the length of the distinguishing sequence is bounded by $n(\frac{n-1}{2})$. Hence the upper bound, as a function only of the number of states, will be:

$$= 2n.\frac{n(n-1)}{2} + 1 + \frac{n(n-1)}{2} - 1 + (n-1).(n-2)$$

$$= n^3 - n^2 + 2n + \frac{n^2}{2} - \frac{n}{2} - 1 + n^2 - 3n + 2$$

$$= n^3 + \frac{n^2}{2} - \frac{3n}{2} + 1$$

Fortunately this bound is much greater than the checking experiment length usually obtained. The length is generally quite close to the following lower bound.

(b) Lower Bound:

The lower bound will be achieved when no transfer sequences are required. With maximum overlap the length of the α-sequence becomes $n + L$. The β-sequence will have a length of $n.(L+1)$.

$\therefore$ LOWER BOUND $= n + L + n.(L+1)$

If sufficient additional outputs are permitted to reduce the DS to one symbol the absolute lower bound becomes $3n+1$.

CONCLUSIONS

In this paper a systematic procedure has been described for modifying a sequential switching circuit in order that it should have short homogeneous distinguishing sequences, and for designing near minimal fault checking experiments. Elements of graph theory have been considered as the preferred vehicle by which the procedure can be programmed on a computer. Some specific classes of machines have been considered, namely those with reset or cyclic permutation columns. It is clear that the techniques employed in these special cases can be applied when only some subset of all of the states has the reset or cyclic permutation property.

The overall procedure is very simple to apply by hand for a small sequential machine since the graph analysis can be done by inspection. If trial and error methods are avoided completely, in most cases the length of the resulting checking experiment will still be close to minimal. However, the designer has the choice of including exhaustive testing in the implementation, in which case a minimal experiment will be obtained. For large machines this latter process would be unacceptably time consuming. One solution to this problem is to have an interactive program in which the graphs are modified by inspection where possible, thereby eliminating some of the more complex matrix techniques. The ordering of specific tasks such as branch elimination, subgraph selection, choice of starting states, etc., presents a problem which is analagous to the prime implicant problem and is unsolved.

The questions left unanswered in this investigation are either related to a very small subset of the sequential machines or are likely to contribute only nominally towards optimizing the procedure. However, an attempt has been made to at least note all of the aspects of the subject that remain to be examined or that should be considered when the procedure is implemented. These include:

(1) determining how the order in which many of the branch elimination aspects may be defined such that the whole process is optimized;

(2) obtaining output assignment algorithms for all machines containing cyclic permutation columns;

(3) selecting suitable start states for the checking experiment and determining the order in which source states should be chosen.

It is felt that a computer implementation of the methods described will lead to at least partial solutions to some of these problems, and that it may be possible to derive some heuristic functions to aid the optimization process.

An upper bound on the experiment length has been calculated which is lower than any previously obtained. However, this bound is still a gross overestimate of the length usually possible. In most cases values close to the lower bound are more probable. This lower bound is the smallest obtainable for a given homogeneous distinguishing sequence under the design philosophy employed in this paper.

BIBLIOGRAPHY

1. Z. Kohavi and P. Lavallee, "Design of sequential machines with fault-detection capabilities," IEEE Trans. Electron. Comput., vol. EC-16, pp. 473-484, Aug. 1967.

2. J.R. Kane and S.S. Yau, "On the design of easily testable machines," Proc. IEEE 12th Annual Symp. Switching and Automata Theory, pp. 38-42, Oct. 1971.

3. G. Gonenc, "A method for the design of fault detection experiments," IEEE Trans. Comp., Vol. C-19, pp. 551-558, June 1970.

4. A.R. Smith, "General shift register sequences of arbitrary cycle length," IEEE Trans. Comp., vol. C-20, pp. 456-459, April 1971.

5. F. Harary, R. Norman and D. Cartwright, Structural Models - An Introduction to the Theory of Directed Graphs, John Wiley and Sons, Inc., 1965.

6. Z. Kohavi, J.A. Rivierre, and I. Kohavi, "Checking experiments for sequential machines," Information Sciences, vol. 7, No. 1, pp. 11-28, Jan. 1974.

7. S. Washall, "A Theorem on Boolean Matrices," J.ACM 9, pp. 11-12, Jan. 1962.

APPENDIX

APPENDIX

The procedure for finding a minimal covering of a directed graph (page 58) requires that the elimination of bridge lines be avoided where possible. It is therefore necessary to be able to recognize bridge lines and to be able to determine whether an alternate path from the node at the beginning of the bridge exists. Recognition of bridge lines using matrix techniques is well known and is discussed in detail in Harary (5). A summary of the segments of that discussion relevant to this application is presented below.

To determine whether an alternative path exists at any point it is necessary to keep an updated record of only the current in-degree and out-degree of each node. These quantities are readily obtained from the adjacency matrix A(D) of the original digraph. The row sums of this matrix are the out-degrees and the column sums the in-degrees of the respective nodes. When the initial node of a bridge is reached, if that node has out-degree greater than one then clearly an alternate path exists. These node degrees also enable the easy recognition of source states, and states which belong to the subset F referred to in Chapter VI.

The principle steps in the process for identifying bridge lines are now presented.

1. Compute the connectedness matrix C(D) of the digraph D. The procedure for this was given in chapter III. Recall that the matrix C(D) gives complete information about the connectedness

category of the components of D. If the $(i,j)^{\underline{th}}$ entry of C(D) is 3 or 2 then the nodes V_i and V_j are in the same strong or weak component respectively. If the entry is zero, V_i and V_j are not connected. The lines in strong components can be ignored for the remainder of the analysis since no strong component can contain a bridge.

2. A start state is chosen and the weak component (D_1) containing this state is determined from C(D).

3. To determine whether a line $x \in D_1$ is a bridge we obtain $A(D_1)$ and from this $A(D_1-x)$. (The entry of 1 in $A(D_1)$ which reflects the presence of x becomes a 0 in $A(D_1-x)$.)

4. Derive $C(D_1-x)$. If the minimum entry of $C(D_1-x)$ is zero then x is a brige.

In programming the design of the checking experiment the procedure described above is probably the least satisfactory aspect of the entire process. Every line of every weak component must be subjected to analysis. In addition, after each path has been established it is necessary to recompute all strong and weak components of the digraph D. Although considerable reduction in the amount of computation can be achieved with sparse matrix techniques and other shortcuts it is arguable that the time involved in avoiding the elimination of bridges is not justified by the potential reduction in the length of the experiment.