AN ALGORITHM FOR SOLVING THE SEQUENTIAL MACHINE IDENTIFICATION PROGRAM

Larry A. Litteral

Ohio State University

Prepared for:

Air Force Office of Scientific Research

June 1973

AN ALGORITHM FOR SOLVING

THE SEQUENTIAL MACHINE IDENTIFICATION PROBLEM

A Thesis

Presented in Partial Fulfillment of the Requirements
for the Degree Master of Science

by

Larry A. Litteral, B.S.

The Ohio State University
1973

Approved by:

Kenneth S. Brace

Advisor
Department of Electrical Engineering

ii

ACKNOWLEDGEMENTS

I wish to thank Professor Kenneth J. Breeding for his unfailing help in the preparation of this paper and also for his guidance in related courses. I am also grateful to Professor Hooshang Hemami for his helpful suggestions and careful review of this work.

My deepest thanks go to my lovely wife for her patience and encouragement throughout my education and without whose help this paper would not have been completed. I would also like to thank my dog, Struedel, without whose help this paper could have been finished months ago.

This research was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-71-2048.

TABLE OF CONTENTS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-773 774

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR - TR - 74 - 0092	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): AN ALGORITHM FOR SOLVING THE SEQUENTIAL MACHINE IDENTIFICATION PROBLEM	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Larry A. Litteral	8. CONTRACT OR GRANT NUMBER(s): AFOSR 71--2048
9. PERFORMING ORGANIZATION NAME AND ADDRESS: The Ohio State University, Department of Electrical Engineering, Columbus, Ohio 43210	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F, 9769-02
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research (NM), 1400 Wilson Blvd, Arlington, Virginia 22209	12. REPORT DATE: June 1973
	13. NUMBER OF PAGES: ~~111~~ 119
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Machine identification
Test sequence
Fault detection
Sequential machine

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Given an unknown sequential machine, the machine identification problem requires that a state table be found for the sequential circuit. An algorithm for handling the machine's identification problem when the machine is strongly connected, is described along with corresponding computer programs. The algorithm proceeds by first determining a lower bound on the number of states. Then an input sequence is applied in such a way as to fill the state table entries. If a contradiction occurs, the table is expanded and the procedure continues. Otherwise, a fault detection experiment is performed to determine if the derived

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20. Abstract (Continued)

state table corresponds to the unknown machine. A number of examples are given.

LIST OF FIGURES

LIST OF FIGURES (Continued)

CHAPTER I

INTRODUCTION

Sequential machine identification is a challenging and complex procedure. As machine size increases, the difficulty of this procedure increases very rapidly. This makes solutions to practical problems totally unfeasible. However, with the speed and accuracy that computers allow, practical machine identification problems are solvable.

Up until the present day, very little work has been done on the machine identification problem. Booth[1] proposed a method where an upper bound b on the number of states in the machine is known. Then all possible b-state machines or less are constructed. Input sequences are then chosen in a manner which eliminates machines possibilities. The process of designing and applying input sequences continues until only one machine remains which must be the machine in question. It is clearly seen that if the upper bound b is large, this solution is not practical.

The topic of this thesis is to present an algorithm to be used as a practical solution to the machine identification problem. Chapter II provides a general description of the operation of the algorithm. Chapter III lists the algorithm along with an example to help the reader understand its operation. A computer program was written to implement the algorithm and Chapter IV discusses in detail the limitations imposed by the program, how to operate the program, and how the program functions. Chapter V is a summary which discusses the major

features of this thesis and some possible areas of further investigation. The computer programs described in Chapter IV and other examples are listed in Appendix I.

CHAPTER II

RATIONALE AND GENERAL DESCRIPTION

2.1. Introduction

The problem of identifying an unknown machine can be stated as follows: Suppose that a machine M whose behavior is to be determined is given to the experimenter as a "black box." The experimenter's objective is to find from the input-output sequences a state table of the given machine.

The following definitions will be helpful in understanding the following sections:[1.]

Definition 2.1.1: A "sequential machine" S (Moore model) is represented by a 5-tuple $\langle X, Q, Z, \delta, \omega \rangle$ where $X = \{x_1, x_2, \ldots x_n\}$ is the set of input symbols, $Q = \{q_1, q_2, \ldots q_m\}$ is the set of states, $Z = \{z_1, z_2, \ldots z_\ell\}$ is the set of output symbols, δ is a mapping $X \times Q$ into Q and is termed the next state mapping, and ω is a mapping of Q onto Z and is termed the output mapping.

Definition 2.1.2: A machine with a state set Q is said to be "strongly connected" if for any pair of states

---

[1.] The notation and definitions used throughout this paper conform as closely as possible to that of Booth [1].

q_j, $q_k \in Q$ there exists at least one input sequence J such that $\delta(J, q_j) = q_k$ where $\delta(J, q_j)$ is understood to be the state in which the machine ends when started in state q_j and input sequence J is applied.

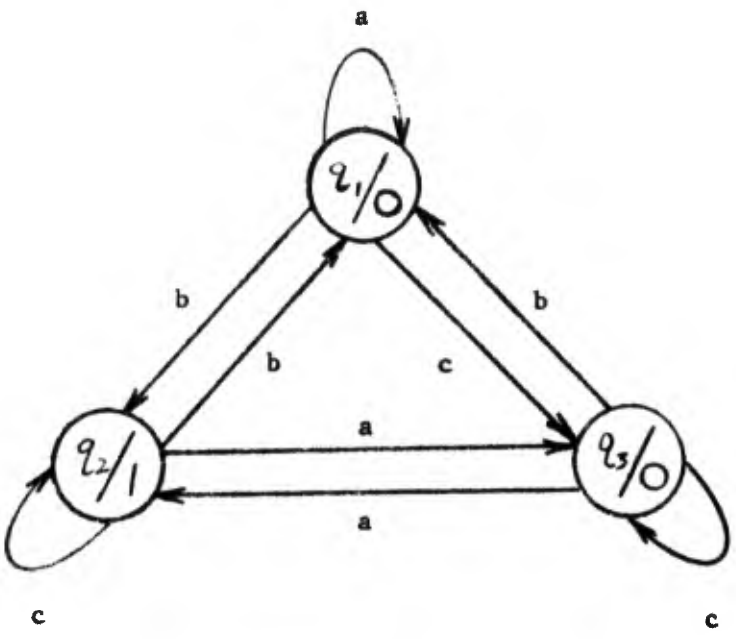
Definition 2.1.3: A "synchronizing sequence" of a machine M is a sequence which takes M to a particular final state, regardless of the output or the initial state. (Note: Some machines possess such sequences; others do not.)

Definition 2.1.4: An input sequence X_0 is said to be a "distinguishing sequence" if the output sequence produced by M in response to X_0 is different for each initial state.

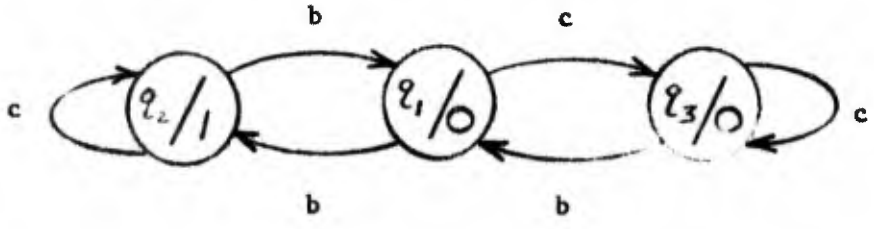
The identification problem as stated above has a solution only if the machine M being tested is strongly connected and the input set X is defined.

Machines which are not strongly connected have a state q_j such that an input sequence does not exist which takes M from q_i to q_j. Thus, it is possible that q_j will not appear in the derived state table since it could be missed.

The input set X must be defined or we cannot completely define the two mappings $\delta(X \times Q)$ and $\omega(Q)$. For example, suppose that we were working with the machine shown in Figure 2.1.1a. The true input set is $X = \{a, b, c\}$. However, assume that we are only told that the input set contained the two elements b and c. Then any



(a)



(b)

Figure 2.1.1. Results of a measurement with an incompletely specified input set.

experiment we perform on the machine would only allow us to determine at most the representation shown in Figure 2.1.1b. As we see from this figure, the lack of knowledge that the symbol a is also an input prevents us from obtaining a complete description of the machine.

In the following sections a general description of and justification for the algorithm introduced in this thesis will be discussed.

2.2. Algorithm Description

The algorithm presented in this paper is designed to derive a state table representation of an unknown Moore machine. However, a Moore to Mealy transformation is a relatively simple, well defined procedure should the experimenter desire a Mealy representation [2].

The algorithm begins by finding a lower bound on the number of states in the unknown machine M. This can be done by observing the following: Let the input to an n-state machine be an arbitrarily long sequence of x_i's. Now, if we let the sequence be long enough so that it is longer than n, the machine must eventually arrive at a state it has previously been in. And, consequently, from this point on, and because the input remains the same, the machine must continue in a periodically repeating fashion. Clearly, for an n-state machine, the period cannot exceed n, and could be smaller.

First, an input sequence of x_i's is applied to M until a periodically repeating output sequence is observed. This procedure is repeated until all input symbols have been used. One cycle of each output sequence corresponding to each of the different input sequences is observed. Let ψ_0 be the largest number of appearances of a single output symbol in one cycle of any of the different output sequences produced by the different input sequences of x_i's.

For example, suppose the set X = {0, 1} is the input of an unknown machine.

Suppose the output sequence produced by a sequence of 0's is

1 0 1 0 1 0 1 0 . . . .

One cycle would be 1 0.

Now, suppose the output sequence produced by a sequence of 1's is

1 0 1 1 1 0 1 1 1 0 1 1 . . . .

One cycle would be 1 0 1 1.

The output symbol "1" appears three times in the cycle 1011 produced by an input sequence of 1's. Thus $\psi_0 = 3$. Initially, a state table containing $\psi_0 z_0$-states, $\psi_0 z_1$-states, . . . , and $\psi_0 z_\ell$-states is made available. Consequently, $\ell\psi_0$ is an upper bound on the number of states in the initial state table. This decreases the number of iterations required when deriving the state table as will be shown later.

The algorithm then develops an input sequence which will fill any unspecified entries in the state table. An input sequence is applied to M and to the state table and entries are made in the state table when required. M's output sequence is compared to the state table's output sequence to detect any errors in the state table. When an error is detected, an entry in the state table is changed and the experiment continued. Corrections are made in a manner so that all machines which have $\ell\psi_0$ states or less are tried. If none of these machines are found to be correct, then M must be larger

than an $\ell\psi_0$-state machine. Thus, a z_0-state, a z_1-state, . . . , and a z_ℓ-state are added to the state table and all $(\ell\psi_0 + \ell)$-state machines or less are tried. This procedure is continued until a completely specified state table is derived.

Once a completely specified state table is derived, it becomes necessary to check this table to verify that it is indeed an equivalent state table representation of the unknown machine. This is done by designing a fault-detection experiment using the state table and then applying the fault detection experiment to both M and the state table. If an error is found, the algorithm returns to its search for an equivalent state table.[2]

In the procedure we shall use, each fault-detection experiment consists of two parts:

1. The first part is an "adaptive" experiment whose aim is to transfer the machine into a particular state, which is the initial state for the second part of the experiment.
2. The second part is a "preset" experiment in which the machine is taken through all possible transitions.

In designing the adaptive part of the experiment, we must first verify that the initial state is indeed the state it is supposed to be. This can be accomplished by displaying the response of each state to the same distinguishing sequence.

---

[2]. The ideas used here are discussed in depth by Kohavi [2].

For example, consider the machine M_3, whose state table is given in Figure 2.2.1. and whose response to the sequences 00 and 01 are summarized in Figure 2.2.2. Suppose that the preset part of the experiment is designed so that state A is the initial state, to which it is therefore necessary to transfer the machine.

First, we apply the sequence 10 and observe the response.

1. If the response is 00, the machine is in state A;
2. If it is 10, apply the transfer sequence T(B, A) = 0;[3]
3. If it is 11, apply T(D, A) = 00.

Now that we have supposedly put the machine in state A, we must verify this. We now apply the distinguishing sequence X_0 = 00. If the machine has operated correctly up to this point, its output is 11, and it is now in state D. Again apply 00, and as a result the machine returns to state A with a response of 00. Now apply T(A, B) = 1. The machine should now be in state B with a response of 0. Now apply the distinguishing sequence 00. This leaves the machine in state C. Again apply 00 and the machine is now in state B, as shown below:

Input:	0	0	0	0	1	0	0	0	0
State:	A		D		A	B		C	B
Output:	0	1	1	0	0	0	0	1	1	0

[3]. Note that a transfer sequence $T(q_i, q_j)$ is the shortest input sequence that takes C machine from state q_i to state q_j.

PS	NS		z
	x = 0	x = 1	
A	C	B	0
B	A	B	0
C	D	D	1
D	B	C	1

Figure 2.2.1. Machine M_3.

Initial State	Response to 00	Final State
A	11	D
B	01	C
C	10	B
D	00	A

(a)

Initial State	Response to 10	Final State
A	00	A
B	00	A
C	10	B
D	11	D

(b)

Figure 2.2.2. Responses of machine M_3.

From the preceding sequences, we note four different responses to the input sequence 00. This verifies that the machine has at least four distinct states.

To complete the experiment, it is now necessary to verify every state transition. The procedure followed here is to apply the input symbol which causes the desired transition and to identify it by applying the distinguishing sequence. Since the machine is in state B, we shall start by applying an input 0, followed by the distinguishing sequence 00. This input sequence takes the machine to state D. Again apply 000 and the machine should now be in state C. This is repeated until all transitions have been verified.

Notice also that the input sequence 100101 is a synchronizing sequence for machine M_3. Thus another fault-detection experiment would be to apply the synchronizing sequence followed by a particular sequence which transfers the machine to a state from which a transition is to be verified. Then the proper input to cause the desired transition is applied. Then the above steps of synchronizing sequence, transfer sequence, and a proper input to cause the desired transition, is repeated until all possible transitions have been verified. In this method the inputs are not dependent upon the preceding outputs as in the adaptive part of the preceding fault-detection experiment. This method lends easily to computer programming.

2.3. Summary

In the previous sections a general discussion was given of the algorithm defined in this thesis. The algorithm searches for a state table which duplicates the input-output sequences of the unknown machine. Fault-detection experiments are used to determine the correctness of a state table once it is derived.

It was also shown that a solution to the machine identification problem exists only if the unknown machine is strongly connected and the input set X is defined.

CHAPTER III

AN ALGORITHM

3.1. Introduction

The algorithm of this chapter presents a detailed procedure for deriving an equivalent state table representation of an unknown Moore machine. The restrictions which apply in this solution have been outlined in Chapter II.

The following definitions are helpful in understanding the mechanics of the algorithm:

Definition 3.1.1: χ is a sequence of input symbols.

$$\chi = x_{i_1} x_{i_2} \cdots x_{i_\ell} \; ; \; x_{i_j} \in X \; .$$

Definition 3.1.2: χ_i is an input sequence in which the only input symbol is x_i ; $x_i \in X$.

Definition 3.1.3: ζ is a sequence of output symbols. $\zeta = z_{i_1} z_{i_2} \cdots z_{i_\ell}$; $z_{i_j} \in Z$.

Definition 3.1.4: ζ_i is a sequence of output symbols which are produced by χ_i.

Definition 3.1.5: In the application of χ_i, the output sequence ζ_i will start to repeat periodically. One cycle in this periodically repeating output sequence will be called ζ_i^*.

Definition 3.1.6: The unknown machine will be referred to as M_1.

Definition 3.1.7: The state table constructed by the algorithm will be referred to as M_2.

Definition 3.1.8: The number of states whose corresponding output is ζ_i in machine M_2 at the j^{th} iteration of the algorithm will be called $\psi_j^{(i)}$. Recall from the discussion in Section 2.2 that the number of ζ_i-states equals the number of ζ_k-states for $i \neq k$. Therefore $\psi_j^{(i)} = \psi_j^{(k)}$ for $i \neq k$ and therefore we will denote $\psi_j^{(i)}$ as ψ_j.

Definition 3.1.9: An indicated error is an incorrect output from the state table M_2 which conflicts with the actual output from the unknown machine M_1.

Definition 3.1.10: The input sequence developed by the algorithm below and the corresponding output sequence from M_1 will be called a "test sequence". The test sequence consists of $\chi_0\ \chi_1 \ldots \chi_n$ followed by a sequence of fault-detection experiments plus any particular inputs requested by the algorithm, along with the corresponding outputs $\zeta_0\ \zeta_1 \ldots \zeta_n$ followed by the corresponding outputs of the fault-detection experiments and other inputs requested by the algorithm.

Note: At the beginning of the algorithm, there is no test sequence. As the algorithm progresses, the test sequence is developed. A test sequence for the example of Section 3.3 is developed in Figure 3.3.2.

3.2. The Algorithm

Based on the discussion of Chapter II and the definitions of Section 3.1 the following algorithm is presented. In the algorithm, the final machine M_2 is derived using all of the applied inputs.

1. Apply χ_0 to M_1 until the output sequence begins to repeat periodically. Apply χ_1 to M_1 until the output sequence begins to repeat periodically. Continue to apply input sequences χ_i until all input symbols have been used. Let ψ_0 be the largest number of appearances of any single output symbol in one cycle of any of the different output sequences that is the ζ_i's (Section 2.2 provided an example of this idea). Let the longest ζ_i which contains ψ_0 appearances of the single output symbol be ζ_i^*. The first symbol of ζ_i^* will be taken as the starting point of the test sequence and the index PIO will have the value 0 at this point. PIO will represent the position in the test sequence during execution of the algorithm. Let L be the length of ζ_i^*. Construct a table as shown in Figure 3.2.2. Note: initially the output set Z is not known. However, as the output sequence of M_1 is observed, the elements of Z become known. Construct a table of entries as shown in Figure 3.2.3. Let the present state of M_2 be q_1 or q_2 or . . . or q_ℓ depending on whether z(PIO) is z_1 or z_2 or . . . or z_ℓ respectively. Increment PIO. Let P = 0 where P is an index which indicates the order in which entries are made in the state table, that is $q_i(P)$. Apply the next L - 1 inputs to M_2 from the test sequence, incrementing PIO after each input. Enter the next state entries, $q_i(0)$, in M_2, where q_i is a previously unused

P I O	Input	Output
		1
	0	1
0	0	0
1	0	0
2	0	0
3	0	0
4	1	1
5	1	0
6	1	1
7	1	0
8	1	1
9	1	0
.	.	.
.	.	.
.	.	.
.	.	.

Figure 3.2.1. An example of a test sequence.

Present State	Next State				Output
q_i	$x = x_1$	$x = x_2$	. .	$x = x_n$	z
q_1					z_1
q_2					z_2
.					.
.					.
q_i					z_ℓ
q_{i+1}					z_1
.					.
.					.
.					.
.					.
q_j					z_ℓ
.					.
.					.
.					.
.					.
.					.
.					.
q_k					z_1
q_{k+1}					z_2
.					.
.					.
.					.
.					.
$q_{\ell\psi_j}$					z_ℓ

Figure 3.2.2. A table used in deriving a state table M_2.

P	Next State $q_i(P)$	P I O
1		
2		
3		
.		
.		
.		
.		
.		
$4\ell\psi_j$		

Figure 3.2.3. A table of entries.

state. This causes M_2 to contain the output cycle ζ_i^*. After the preceding instructions have been completed, let P = 1 and go to Step 2.

2. Continue to apply the test sequence to M_2. Each time an entry is to be made in M_2, enter $q_1(P)$ or $q_2(P)$ or . . . or $q_\ell(P)$ in the state table, (Figure 3.2.2), if the corresponding output is z_1 or z_2 or . . . or z_ℓ, respectively. Also enter the state $q_i(P)$ and the corresponding PIO value in the table of entries, Figure 3.2.3., and increment P. If an indicated error is encountered, go to Step 4, otherwise go to Step 3.

3. First look at all the rows of M_2 except the PS. If all these rows are either completely filled or completely empty and the PS is completely filled go to Step 7. Otherwise, if the PS is completely filled, by looking at M_2 obtain an input sequence which will cause M_2 to transition to a state which contains at least one entry but not completely filled. Apply this input sequence to M_1 and M_2. If the PS contains a blank entry or after transitioning M_2 to a state which has a blank entry, apply the proper input, to fill the blank entry, to M_1 and M_2. Again enter the proper $q_i(P)$ as explained in Step 2 in the state table, Figure 3.2.2, and in the table of entries Figure 3.2.3 along with the corresponding PIO value. Increment P. If an indicated error is encountered, go to Step 4. Otherwise repeat Step 3.

4. Let P = P - 1. Let the present state of M_2 be that which contains $q_i(P)$ corresponding to the present P value as an entry. Let PIO be that which is listed in the table of entries Figure 3.2.3

corresponding to $q_i(P)$. Using i corresponding to the same $q_i(P)$, if $i + \ell > \ell\psi_j$ where ℓ is the number of output symbols in Z, go to Step 5. Otherwise change $q_i(P)$ to $q_{i+\ell}(P)$ in M_2 and in the table of entries. Let the present state of M_2 be $q_{i+\ell}$. Increment P and PIO and go to Step 2.

5. If P = 1 go to Step 6. If not, delete $q_i(P)$ from M_2 and delete the corresponding entry from the table of entries. Go to Step 4.

6. Let $\psi_{j+1} = \psi_j + 1$. Make appropriate modifications to the state table and to the table of entries. Delete $q_i(P)$ from M_2 and the corresponding entry from the table of entries and go to Step 2.

7. Derive a fault-detection experiment using M_2. Apply the experiment one input at a time to M_1 and M_2. If an indicated error is encountered, terminate the fault-detection experiment and go to Step 4. If not and the end of the experiment is reached, the state table M_2 is an equivalent state table representation of the unknown machine M_1.

Recall from the discussion in Section 2.2 that in searching for a state table representation of the unknown machines, the algorithm generates all state tables of $\ell\psi_j$ states or less. After each iteration the value ψ_j is increased by one, that is $\psi_{j+1} = \psi_j + 1$, where $j = 0, 1, 2, \ldots$, and the search is continued.[4] The algorithm does not impose an upper bound on ψ_j. Therefore the algorithm must converge on an equivalent state table representation of the unknown machine.

---

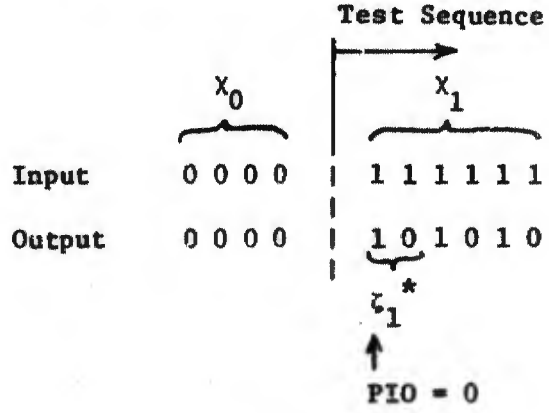
[4] An iteration is understood to mean each time the value ψ_j is changed.

3.3. An Example

At first glance the algorithm will seem very long and complex. The following example will help clarify the procedure:

Example 1: Find a state table M_2, which is an equivalent state table representation of the machine shown in Figure 3.3.1. It is given that X = {0, 1}, and assume that the machine is in state A.

Step 1. This step corresponds to Step 1 of the algorithm. A record of the test sequence, the present states of M_1 and M_2 and PIO will be maintained in Figure 3.3.2. Apply a sequence of zeros χ_0 to M_1 until the output appears to be repeating periodically. The first four inputs yield ζ_0 = 0000. The output appears to be a sequence of repeating zeros. Thus ζ_0^* = 0. Here we terminate the input sequence χ_0. Now apply a sequence of ones χ_1 to M_1 until the output appears to be repeating periodically. After six inputs of χ_1 the output sequence yielded is ζ = 101010. The output appears to be a sequence where the subsequence 10 is repeating. Thus ζ_1^* = 10. Now terminate the input sequence χ_1. At this point the test sequence is:



PS	NS		z
	x = 0	x = 1	
A	A	B	0
B	C	A	1
C	B	B	1

Figure 3.3.1. A machine

P I O	Input	PS of M_1	Output	PS of M_2
		A	0	
	0	A	0	
	0	A	0	
	0	A	0	
	0	A	0	
0	1	B	1	q_2
1	1	A	0	q_1
2	1	B	1	q_2
3	1	A	0	q_1
4	1	B	1	q_2
5	1	A	0	q_1
6	0	A	0	q_1
7	1	B	1	q_2
8	0	C	1	q_4
9	1	B	1	q_2
10	0	C	1	q_4
11	0	B	1	q_2
12	1	A	0	q_1
13	0	A	0	q_1
14	1	B	1	q_2

Figure 3.3.2. A test sequence derived in Example 1 as it appears at the end of this example. (Continued)

Figure 3.3.2. (Continued)

P I O	Input	PS of M_1	Output	PS of M_2
15	0	C	1	q_4
16	1	B	1	q_2
17	0	C	1	q_4
18	1	B	1	q_2
19	1	A	0	q_1
20	1	B	1	q_2
21	0	C	1	q_4
22	1	B	1	q_2
23	1	A	0	q_1
24	0	A	0	q_1
25	1	B	1	q_2
26	0	C	1	q_4
27	1	B	1	q_2
28	1	A	0	q_1
29	1	B	1	q_2
30	1	A	0	q_1
31	0	A	0	q_1
32	1	B	1	q_2
33	0	C	1	q_4

Figure 3.3.2. (Continued)

P I O	Input	PS of M_1	Output	PS of M_2
34	0	B	1	q_2
35	1	A	0	q_1
36	0	A	0	q_1
37	1	B	1	q_2
38	0	C	1	q_4
39	1	B	1	q_2

Consider, now one cycle of ζ_0 and ζ_1. The output symbol 0 appears at most one time in one cycle of each sequence. The output symbol 1 appears at most one time in one cycle of each sequence. Let ψ_0 be the largest number of appearances of any single output symbol in one cycle of any of the different output sequences, that is the ζ_i's. Thus $\psi_0 = 1$. Let the longest ζ_i which contains ψ_0 appearances of the single output symbol be ζ_i^*. Thus $\zeta_i^* = \zeta_1^*$. Let L be the length of ζ_1^*. Thus L = 2. The input symbol which corresponds to the first output symbol ζ_1^* will be taken as the starting point of the test sequence and the index PIO will have the value 0 at this point.

Construct a table as shown in Figure 3.2.2. See Figure 3.3.3. Note: At this point the only output symbols known are the set {0, 1}.

Construct a table of entries as shown in Figure 3.2.3. See Figure 3.3.4.

Let the present state of M_2 be q_2 as z(PIO) = z(0) = 1. Increment PIO. Thus PIO = 1. Let P = 0 where P is an index which indicates the order in which entries are made in the state table, that is $q_i(P)$.

Apply the next L - 1 = 1 input to M_2 from the test sequence. That is the input corresponding to PIO = 1. Enter $q_1(0)$ in the x = 1 column of row q_2 in M_2 as x = 1 for PIO = 1 and z = 0 for PIO = 1. Increment PIO. At this point M_2 is as shown in Figure 3.3.5.

The present state of M_2 now is q_1. Let P = 1 and go to Step 2.

Step 2. This step corresponds to Step 2 of the algorithm. Continue to apply the test sequence to M_2. Each time an entry is to be made in M_2, enter $q_1(P)$, $q_2(P)$, . . . , or $q_\ell(P)$ in M_2 if the

PS	NS		z
	x = 0	x = 1	
q_1			0
q_2			1

Figure 3.3.3. A table used to derive a state table.

P	Next State $q_i(P)$	P I O
1		
2		
3		
4		

Figure 3.3.4. A table of entries used to record entry data.

PS	NS		z
	x = 0	x = 1	
q_1			0
q_2		$q_1(0)$	1

Figure 3.3.5. M_2 at completion of Step 1.

PS	NS		z
	x = 0	x = 1	
q_1		$q_2(1)$	0
q_2		$q_1(0)$	1

Figure 3.3.6. M_2 at completion of Step 2.

corresponding output is z_1, z_2, . . . , or z_ℓ respectively. Upon applying the next input from the test sequence corresponding to PIO = 2 to M_2 an entry is required in M_2. Enter $q_2(1)$ in the x = 1 column of row q_1. M_2 is now as shown in Figure 3.3.6.

Enter $q_2(1)$ and PIO = 2 in the table of entries. The table of entries is now as shown in Figure 3.3.7. Increment P and PIO. Thus P = 2 and PIO = 3. The present state of M_2 is now q_2. Continue to apply the test sequence to M_2. Application of the presently remaining three inputs in the test sequence does not indicate an error and requires no more entries in M_2. Thus PIO = 4, 5, 6 and the present state of M_2 PS = q_1, q_2, q_1. No errors were indicated therefore go to Step 3.

Step 3. This step corresponds to Step 3 of the algorithm. The present state q_1 of M_2 has a blank entry as seen in Figure 3.3.6, and applying a "0" input will fill this entry. Thus apply a "0" input to M_1 and record the results in the test sequence. The corresponding output from M_1 was a "0". Therefore, the proper entry in the x = 0 column of row q_1 of M_2 would be $q_1(2)$. Enter this in M_2 and in the table of entries. The state table M_2 is represented by Figure 3.3.8. The table of entries is now represented by Figure 3.3.9. Increment P. Thus, P = 3. Go to Step 4.

Step 4. This step corresponds to Step 3 of the algorithm. Step 3 of the algorithm is repeated as no errors have been encountered with M_2 in its present form and all the information provided by the test sequence in its present form has been used. The present state of M_2

P	Next State $q_1(P)$	P I O
1	$q_2(1)$	2
2		
3		
4		

Figure 3.3.7. The table of entries at completion of Step 2.

PS	NS		z
	x = 0	x = 1	
q_1	$q_1(2)$	$q_2(1)$	0
q_2		$q_1(0)$	1

Figure 3.3.8. M_2 at completion of Step 3.

P	Next State $q_i(P)$	P I O
1	$q_2(1)$	2
2	$q_1(2)$	6
3		
4		

Figure 3.3.9. The table of entries at completion of Step 3.

is q_1. From M_2 (Figure 3.3.8) we see that the row q_2 has a blank entry. An input "1" will put M_2 in state q_2, and then an input "0" will cause the blank entry in state q_2 to be filled. Thus apply the sequence {10} to M_1 and record the results in the test sequence. The test sequence now is represented by Figure 3.3.2 from the top of the figure to the first dotted line. Apply the sequence {10} to M_2. The output produced by the "1" input corresponds to the output produced by M_1. For the "0" input M_1 produced a "1" output. Thus enter $q_2(3)$ in the $x = 0$ column of M_2 and in the table of entries corresponding to $P = 3$.

M_2 is now represented by Figure 3.3.10. The table of entries is now represented by Figure 3.3.11. Increment P. Thus $P = 4$. The present state of M_2 is q_2. No indicated errors were encountered during execution of this step. Go to Step 5.

Step 5. This step corresponds to Step 3 of the algorithm. By looking at Figure 3.3.10, we see that M_2 is now completely specified. That is, there are no blank entries in M_2. Thus, we have reached the point where a state table has been derived. Now, we must ascertain whether or not this table is an equivalent representation of the machine being tested. Go to Step 6.

Step 6. This step corresponds to Step 7 of the algorithm. Derive a fault-detection experiment using M_2. Using the procedures outlined in Section 2.2 we see that a fault-detection experiment for M_2 is the sequence {11011101}. Begin applying this sequence to M_1 and M_2. If

PS	NS		z
	x = 0	x = 1	
q_1	$q_1(2)$	$q_2(1)$	0
q_2	$q_2(3)$	$q_1(0)$	1

Figure 3.3.10. M_2 at completion of Step 4.

P	Next State $q_1(P)$	P I O
1	$q_2(1)$	2
2	$q_1(2)$	6
3	$q_2(3)$	8
4		

Figure 3.3.11. The table of entries at completion of Step 4.

an indicated error is encountered, terminate the fault-detection experiment. On applying the first element of the experiment, the output produced by M_1 is "1" while the output indicated by M_2 is "0". Thus, M_2 must contain an error. Go to Step 7. Note: The test sequence is now represented by Figure 3.3.2 down to the second dotted line.

Step 7. This step corresponds to Step 4 of the algorithm. Let $P = P - 1$. Thus $P = 3$. Let the present state of M_2 be that which contains $q_1(3)$ as an entry. Thus, let the present state of M_2 be q_2. Let PIO be that which is listed in the table of entries corresponding to $q_1(3)$. See Figure 3.3.11. Thus, let PIO = 8. Using i corresponding to the same $q_1(P)$, $i + \ell > \ell\psi_0$, that is $2 + 2 > 2(1)$, thus go to Step 8. Note: Recall that ℓ is the number of output symbols in the output set Z.

Step 8. This step corresponds to Step 5 of the algorithm. $P = 3$, therefore delete $q_1(3)$ from M_2 and delete the corresponding entry from the table of entries. Thus M_2 and the table of entries are now represented by Figures 3.3.8 and 3.3.9 respectively. Go to Step 9.

Step 9. This step corresponds to Step 4 of the algorithm. Let $P = P - 1 = 2$. Let the present state of M_2 be that which contains $q_1(2)$ as an entry, that is q_1. Let PIO be that which is listed in the table of entries corresponding to $q_1(2)$, that is PIO = 6. Using i corresponding to $q_1(2)$, $i + \ell > \ell\psi_0$, that is $1 + 2 > 2(1)$. Go to Step 10.

Step 10. This step corresponds to Step 5 of the algorithm. P = 2, therefore delete $q_i(2)$ from M_2 and delete the corresponding entry from the table of entries. Thus M_2 and the table of entries are now represented by Figures 3.3.6 and 3.3.7, respectively. Go to Step 11.

Step 11. This step corresponds to Step 4 of the algorithm. Let P = P - 1 = 1. Let the present state of M_2 be that which contains $q_i(1)$ as an entry, that is q_1. Let PIO be that which is listed in the table of entries corresponding to $q_i(1)$, that is PIO = 2. Using i corresponding to $q_i(1)$, $i + \ell > \ell\psi_0$, that is $2 + 2 > 2(1)$. Go to Step 12.

Step 12. This step corresponds to Step 5 of the algorithm. P = 1. Go to Step 13.

Step 13. This step corresponds to Step 6 of the algorithm. Let $\psi_1 = \psi_0 + 1 = 2$. Make appropriate modifications to M_2 and to the table of entries and delete $q_1(1)$ from M_2 and the corresponding entry from the table of entries. M_2 and the table of entries are now represented by Figures 3.3.12 and 3.3.13, respectively. Go to Step 14.

Step 14. This step corresponds to Step 2 of the algorithm. Recall that the present state of M_2 is q_1, PIO = 2, and P = 1. Apply the test sequence one input at a time to M_2. For the input corresponding to PIO = 2, enter $q_2(1)$ in the x = 1 column of row q_1 of M_2. Also, enter $q_2(1)$ and PIO = 2 in the table of entries. M_2 and the table of entries are now represented by Figures 3.3.14 and 3.3.15, respectively.

PS	NS		z
	x = 0	x = 1	
q_1			0
q_2		$q_1(0)$	1
q_3			0
q_4			1

Figure 3.3.12. M_2 at the completion of Step 13.

P	Next State $q_1(p)$	P I O
1		
2		
3		
4		
5		
6		
7		
8		

Figure 3.3.13. The table of entries at completion of Step 13.

PS	NS		z
	x = 0	x = 1	
q_1		$q_2(1)$	0
q_2		$q_1(0)$	1
q_3			0
q_4			1

Figure 3.3.14. M_2 as it appears in part of Step 14.

P	Next State $q_1(P)$	P I O
1	$q_2(1)$	2
2		
3		
4		
5		
6		
7		
8		

Figure 3.3.15. The table of entries as it appears in part of Step 14.

Increment PIO and P thus, PIO = 3 and P = 2. The present state of M_2 is now q_2. In applying the next three inputs from the test sequence, those corresponding to PIO = 3, 4, and 5, no entries are needed and no errors encountered. So we have reached the point where the present state of M_2 is q_1, P = 2, and now we apply the input from the test sequence corresponding to PIO = 6 to M_2. Here we enter $q_1(2)$ in the x = 0 column of row q_1 in M_2. Also enter $q_1(2)$ and PIO = 6 in the table of entries corresponding to P = 2. M_2 and the table of entries are now represented by Figures 3.3.16 and 3.3.17, respectively. The present state of M_2 now is q_1, P = 3, and we apply the input corresponding to PIO = 7 to M_2. No entry is required and no error is encountered. Now the present state of M_2 is q_2 and we apply the input corresponding to PIO = 8 to M_2. Here we enter $q_2(3)$ in the x = 0 column of row q_2 in M_2. Also enter $q_2(3)$ and PIO = 8 in the table of entries corresponding to P = 2. M_2 and the table of entries are now represented by Figures 3.3.18 and 3.3.19, respectively. Now the present state of M_2 is q_2, P = 4, and we apply the input corresponding to PIO = 9 to M_2. An indicated error occurs. Go to Step 15.

Step 15. This step corresponds to Step 4 of the algorithm. Let P = P - 1 = 3. Let the present state of M_2 be that which contains $q_1(3)$ as an entry, that is q_2. Let PIO be that which is listed in the table of entries corresponding to $q_1(3)$, that is let PIO = 8. Using i corresponding to $q_1(3)$, $i + \ell \neq \ell\psi_1$, that is $2 + 2 \neq 2(2)$. Thus, change $q_2(3)$ to $q_4(3)$ in M_2 and in the table of entries. Increment P and PIO. Thus, P = 4 and PIO = 9. Let the present state of M_2

PS	NS		z
	x = 0	x = 1	
q_1	$q_1(2)$	$q_2(1)$	0
q_2		$q_1(0)$	1
q_3			0
q_4			1

Figure 3.3.16. M_2 as it appears in part of Step 14.

P	Next State $q_1(P)$	P I O
1	$q_2(1)$	2
2	$q_1(2)$	6
3		
4		
5		
6		
7		
8		

Figure 3.3.17. The table of entries as it appears in part of Step 14.

PS	NS		z
	x = 0	x = 1	
q_1	$q_1(2)$	$q_2(1)$	0
q_2	$q_2(3)$	$q_1(0)$	1
q_3			0
q_4			1

Figure 3.3.18. M_2 as it appears in part of Step 14.

P	Next State $q_1(P)$	P I O
1	$q_2(1)$	2
2	$q_1(2)$	6
3	$q_2(3)$	8
4		
5		
6		
7		
8		

Figure 3.3.19. The table of entries as it appears in part of Step 14.

be q_4. M_2 and the table of entries are now represented by Figures 3.3.20 and 3.3.21, respectively. So to Step 16.

Step 16. This step corresponds to Step 2 of the algorithm. The present state of M_2 is q_4, P = 4, and we apply the input corresponding to PIO = 9 to M_2. Enter $q_2(4)$ in the x = 1 column of row q_4 in M_2. Also enter $q_2(4)$ and PIO = 9 in the table of entries corresponding to P = 4. M_2 and the table of entries are now represented by Figures 3.3.22 and 3.3.23, respectively. At this point all the information contained so far in the test sequence has been used. The present state of M_2 is q_2 and P = 5. Go to Step 17.

Step 17. This step corresponds to Step 3 of the algorithm. Notice that all the rows of M_2 are either completely filled or completely empty except row q_4. If we can fill the blank entry in row q_4 of M_2 the state table will be completely specified, provided that state q_3 does not appear as a next state entry. From state q_2, the present state of M_2, a "0" input will transition M_2 to state q_4. Then a "0" input will fill the blank entry in row q_4. Thus apply the sequence {00} to M_1 and M_2 and record the results in the test sequence. The first input causes no error. On applying the second input from the above sequence, M_1 produces an output of "1". Thus enter $q_2(5)$ in the x = 0 column of row q_4 in M_2 and enter $q_2(5)$ and PIO = 11 in the table of entries. The present state of M_2 now is q_2, P = 6, and PIO = 12. The test sequence is now represented by Figure 3.3.2 from the top to the third dotted line. M_2 and the table of entries are now represented by Figures 3.3.24 and 3.3.25, respectively. Go to Step 18.

PS	NS		z
	x = 0	x = 1	
q_1	$q_1(2)$	$q_2(1)$	0
q_2	$q_4(3)$	$q_1(0)$	1
q_3			0
q_4			1

Figure 3.3.20. M_2 as it appears at completion of Step 15.

P	Next State $q_1(P)$	P I O
1	$q_2(1)$	2
2	$q_1(2)$	6
3	$q_4(3)$	8
4		
5		
6		
7		
8		

Figure 3.3.21. The table of entries as it appears at completion of Step 15.

PS	NS		z
	x = 0	x = 1	
q_1	$q_1(2)$	$q_2(1)$	0
q_2	$q_4(3)$	$q_1(0)$	1
q_3			0
q_4		$q_2(4)$	1

Figure 3.3.22. M_2 as it appears at completion of Step 16.

P	Next State $q_1(P)$	P I O
1	$q_2(1)$	2
2	$q_1(2)$	6
3	$q_4(3)$	8
4	$q_2(4)$	9
5		
6		
7		
8		

Figure 3.3.23. The table of entries as it appears at completion of Step 16.

PS	NS		z
	x = 0	x = 1	
q_1	$q_1(2)$	$q_2(1)$	0
q_2	$q_4(3)$	$q_1(0)$	1
q_3			0
q_4	$q_2(5)$	$q_2(4)$	1

Figure 3.3.24. M_2 as it appears at completion of Step 17.

P	Next State $q_1(P)$	P I O
1	$q_2(1)$	2
2	$q_1(2)$	6
3	$q_4(3)$	8
4	$q_2(4)$	9
5	$q_2(5)$	11
6		
7		
8		

Figure 3.3.25. The table of entries as it appears at completion of Step 17.

Step 18. This step corresponds to Step 3 of the algorithm. All the rows of M_2 are now either completely filled or completely empty and the row corresponding to the present state of M_2 is completely filled. Go to Step 19.

Step 19. This step corresponds to Step 7 of the algorithm. Derive a fault-detection experiment using M_2. Using the procedures outlined in Section 2.2 it is easily found that a fault-detection experiment for M_2 is the sequence

$$\{\ 10101011101101011111010010101\ \}\ .$$

Apply this sequence to M_1 and M_2 one element at a time and record the results in the test sequence. After applying the entire sequence to M_1 and M_2 no errors were encountered. Thus M_2 is a three state machine and is an equivalent representation of the machine being tested. The row q_3 in M_2 is not used, so M_2 as it appears at the end of the algorithm is represented by Figure 3.3.26. A comparison of the following derived table and the actual table being tested (Figure 3.3.1) reveals that the tables are equivalent.

PS	NS		z
	x = 0	x = 1	
q_1	q_1	q_2	0
q_2	q_4	q_1	1
q_4	q_2	q_2	1

Figure 3.3.26. The final state table as derived by the algorithm.

3.4. Computer Example

This section presents a solution to the same example shown in Section 3.3. The program of Appendix I was used in the solution of this example.

The data provided at the end of the example is in octal notation. Incorrect entries refers to those entries made in M_2 which were subsequently changed or deleted by Step 7 of the algorithm. Additional notation on the listing is provided to help understand the information.

00 . . . . . The first column consists of the
00 input sequence applied to M_1.
00
00 . . . . . The second column consists of the
11 corresponding outputs from M_1.
10
11
10
11
10
00
11
01 . . . . . The combined sequences are referred to
in the algorithm as a test sequence.

PS	NS		Z	
	0	1		. . . . . The first state table de-
S 1	S 1(2)	S 2(1)	0	rived by the algorithm.
S 2	S 2(3)	S 1	1	

PS	NS		Z	
	0	1		
S 1	S 1(2)	S 2(1)	0	. . . . . The second state table de-
S 2	S 2(3)	S 1	1	rived by the algorithm.
S 3			0	
S 4			1	

01
01
11

PS	NS		Z	
	0	1		
S 1	S 1(2)	S 2(1)	0	. . . . . The last state table de-
S 2	S 4(3)	S 1	1	rived by the algorithm.
S 3			0	
S 4	S 2(4)	S 2(5)	1	

```
10
00
11
10
00
11
01
11
10
11
10
00
11
01
11  . . . . . A fault detection experiment for checking
01             the preceding table.
11
10
11
01
11
01
01
10
00
11
01
11
```

PS	NS		Z
	0	1	
S 1	S 1(2)	S 2(3)	0
S 2	S 4(3)	S 1	1
S 4	S 2(4)	S 2(5)	1

```
# OF INPUTS      000054
# OF INCORRECT ENTRIES      000004
# OF INPUTS/# OF INCORRECT ENTRIES = 000013 + 000000/000004


END
```

CHAPTER IV

COMPUTER IMPLEMENTATION

4.1. Introduction

Clearly to implement the preceding algorithm which is designed for a general case, as a computer program certain limitations must be put on the machine to be tested. In the case of the program listed in Appendix I, these limitations are:

1. The input set X is restricted to X = {0, 1}.
2. The output set Z is restricted to Z = {0, 1}.
3. The maximum number of states is limited to ten 0-states and ten 1-states for a total of twenty states.
4. The machine being tested must also possess a synchronizing sequence. This is because there are different fault-detection experiments for different classes of machines. Thus we limit our solution to one particular class of machines.

Throughout the operation of the program, the operator will act as the "black box" or machine being tested. After the program is loaded and started, an input will be printed on the teletype by the computer. The operator will then determine the correct output response and type it on the teletype. The computer will echo a line terminator. After an input has been printed by the computer, the operator, before printing the output response, may request the state table in its immediate form be printed. To do this the operator types a "P". The computer will accept this and then wait for the operator to print the output for the preceding input. The operator then prints the output

symbol and the computer will terminate the line and print the state table in its immediate form. After the program has a completely specified table constructed, the state table will be printed before a fault-detection experiment is initiated. After finding an equivalent state table for the "black box", the program will again print the state table leaving out any unused states. This will be followed by a data list in octal notation, the total number of inputs applied, the total number of incorrect entries made, and a ratio of these two values.

The computer used was the PDP-9 of the Department of Electrical Engineering, Ohio State University. All programs were assembled and loaded under the Digital Equipment Company's PDP-9 disk operating system KM9-15 V5A.

In the listing of Appendix I, all numbers contained in the program instructions are in octal notation while all numbers in the comments section are listed in decimal notation.

4.2. Main Program

The START routine is the controlling routine for the program. A general flow chart for this routine is provided in Appendix I. Detailed comments are listed in the program. Below is a block diagram which describes the calling sequence of the various routines in the program. Blocks call other blocks to their right, while returns are performed to the left. See Figure 4.2.1.

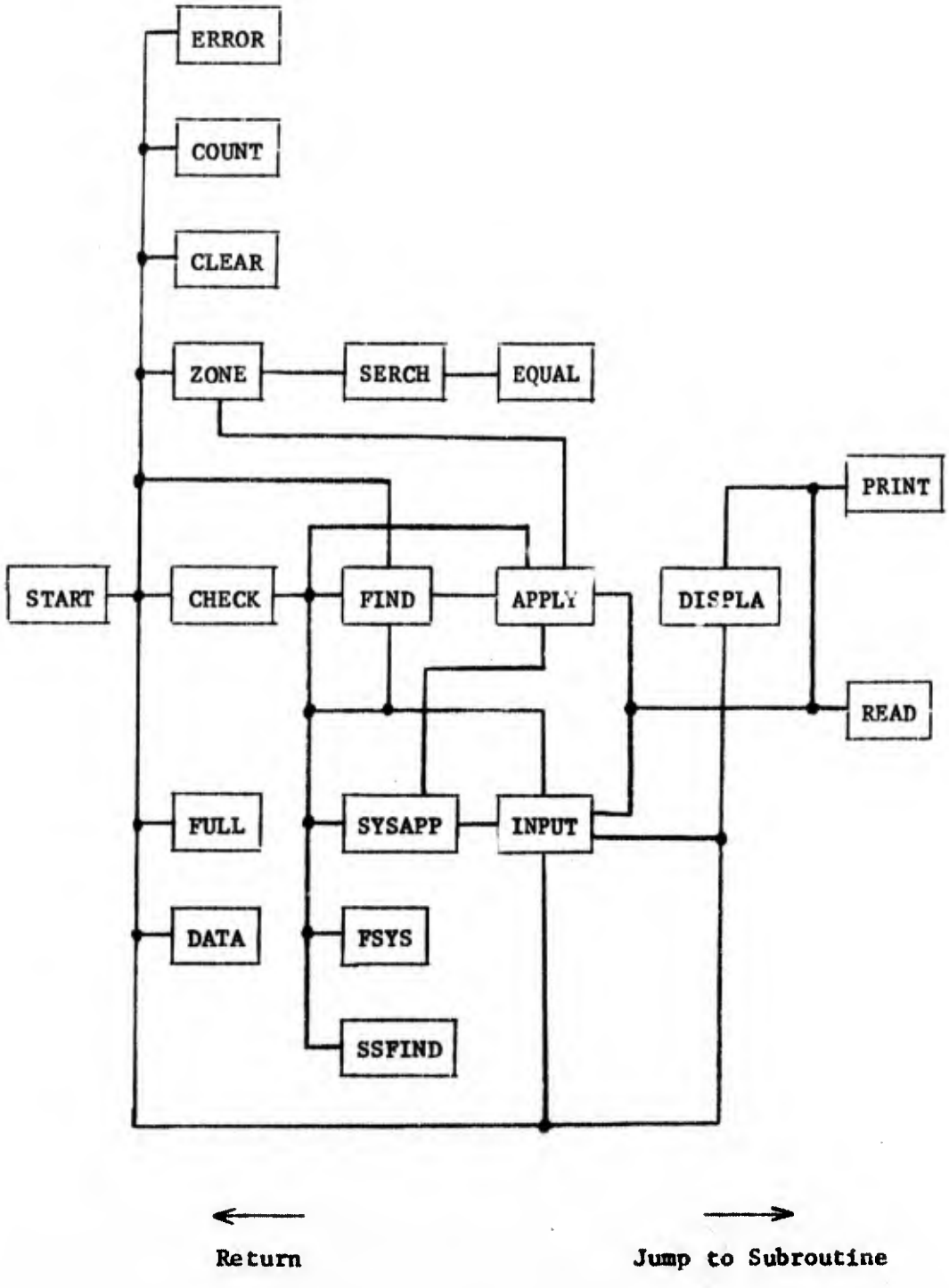


Figure 4.2.1. A block diagram of the program call sequence.

4.3. APPLY Routine

APPLY stores the input in the test sequence. Then when a character is typed on the teletype, it is decoded. If it is a proper output character, that is a "0" or a "1", it is stored in the test sequence and return to calling program is performed. If the character is a "P", a display flag is set and the routine then waits for the proper output. If any other character is received, a "?" is printed on the teletype and the routine waits for a proper output.

4.4. ZONE Routine

Zone applies 0 inputs until three cycles of a repeating output sequence are recognized. Then 1 inputs are applied until three cycles of a repeating output sequence are again recognized.

4.5. SERCH Routine

SERCH looks at the output sequence produced by an input sequence of 0's and searches for three cycles of a repeating output sequence. When three cycles are observed control is returned to ZONE which begins application of an input sequence of 1's. The SERCH routine is then called to signal when three cycles of a repeating output sequence is found which corresponds to an input sequence of 1's.

SERCH is called by ZONE after each application of an input.

4.6. COUNT Routine

COUNT takes one cycle of a repeating output sequence and counts the number of 0 outputs and the number of 1 outputs. The larger of

these two numbers is a lower bound on the number of states in the state table as constructed by the program.

4.7. INPUT Routine

The state table is stored using four words of core for each row in the state table. The first and second words contain the next state for a 0 input and its P value respectively. The third and fourth words contain the next state for a 1 input and its P value, respectively. The first four words are used for the first state of the state table as shown in Figure 3.1.2. The next four words are used for the second state of the state table and so on.

INPUT gets the next input from the test sequence and looks at the proper entry in the state table. If a next state is specified, INPUT compares the next state's output with the next output from the test sequence. If there is an error, an error flag is set and the routine is exited. If there is no error, the present state is updated and the routine is repeated. If no next state is specified, INPUT enters a next state in the state table, updates the present state and continues. If INPUT reaches the end of the test sequence, the routine is exited.

4.8. FIND Routine

FITAB is the storage area used to find an input sequence which will cause the state table to transition from q_i to q_j. The first word of FITAB is used to store q_i. The next word will contain the first input. The third word will contain the state implicated by q_i.

The words of FITAB continue to alternate with an input and a next state. All input sequences of length $\ell\psi_j - 1$ or less are tried. When an input sequence is found which will cause a transition in the state table from q_i to q_j, the sequence is stored in FITAB2 with eighteen inputs per word. The routine continues to search for a shorter sequence. When the shortest sequence has been found, the routine applies the input sequence to the "black box" then exits.

4.9. SSFIND Routine

This routine is a subsection of FIND and is called only by CHECK. CHECK specifies a state q_j to which the state table is to transition. SSFIND obtains the shortest input sequence to do this, applies it to the "black box" and exits.

4.10. ERROR Routine

This routine changes the last entry that was made in the state table from q_i to q_{i+2}. It then checks to see if q_{i+2} is outside the table. If it is, q_{i+2} is deleted from the table and the next to the last entry that was made in the state table is changed from q_i to q_{i+2}, and so on. If q_{i+2} is within the table, the routine is exited. After all possible states have been tried in each entry, a 0 output state and a 1 output state are added to the table and the routine is exited.

4.11. FULL Routine

This routine looks at each row in the state table. If no row contains "only" one blank next state entry, the table is considered completely specified and a flag is set. A row which has no entries is not used by the state table and can be considered nonexistent.

4.12. CHECK Routine

This routine generates the fault-detection experiment to check the state table. A check flag is set to indicate that CHECK is the controlling routine. If an error in the state table is found, the check flag is cleared and the routine is exited.

4.13. SYSAPP Routine

This routine takes the synchronizing sequence generated by FSYS and applies it to the "black box". The sequence is stored with eighteen inputs per word in the storage location SYNT2.

4.14. FSYS Routine

Storage area SYNT1 is used to derive the synchronizing sequence for a state table. Two words are used for each node in a branch of the synchronizing tree. One branch at a time is developed. The high order bit of the first word is the input which produces that node of the tree. The low order two bits of the first word and the entire eighteen bits of the second word are used for the node itself. One bit represents one state in the state table. State q_1 is represented by the low order bit

of the second word. q_{18} is represented by the high order bit of second word. q_{19} and q_{20} are represented by the two low order bits of the first word. The next two words represent the next node in the branch.

All input sequences are tried. After a synchronizing sequence is found, it is stored in SYNT2 allocating eighteen inputs per word. FSYS then continues to search for a shorter synchronizing sequence. After all input sequences of maximum length or shorter are tried, the routine is exited.

If no synchronizing sequence is found an error flag is set when the routine exits.

A branch in the synchronizing tree terminates if a node appears twice in the same branch. The same branch is specified because storage area size permits working with only one branch at a time. A branch also terminates if the length of the sequence reaches the maximum length. The maximum length as stated previously is $(\ell\psi_j - 1)^2\ell\psi_j/2$ where $\ell\psi_j$ is the number of states in the state table. Also when a synchronizing sequence is found, the maximum length used in the search for a shorter synchronizing sequence is the length of the previously found synchronizing sequence minus one.

4.15. Remaining Routines

The remaining routines are small routines for printing, reading, printing data, printing the state table, etc. The comments contained in these routines will suffice to explain their purpose.

4.16. Summary

A general description of how the program listed in Appendix I is implemented and how each routine is used is given. A flow chart (See Appendix I) is provided to give the reader a basic understanding of the overall function of the program. Also a block diagram was given to provide an understanding of the calling sequence of the program.

CHAPTER V

SUMMARY AND CONCLUSIONS

This thesis presented an algorithm for the derivation of an equivalent state table representation of an unknown machine using only input and output sequences associated with the unknown machine. It was shown that the machine must meet two restrictions. The machine must be strongly connected and its input set must be defined.

A computer program which implements the algorithm was listed and it was shown that other limitations had to be put on the machine mainly because of limited storage area available. The machine was limited to a maximum of twenty states. Even so the routine which finds a synchronizing sequence for the state table required a maximum of 7000 octal locations in memory. As the number of states increases, the size of memory required grows very rapidly.

In designing the FSYS routine it was quickly observed that storing a complete synchronizing tree in a limited amount of memory is practically impossible. A synchronizing sequence may be as long as $(N - 1)^2N/2$ where N is the number of states in the machine. That is to say that the synchronizing tree could have $(N - 1)^2N/2$ levels and Y nodes where Y is:

$$Y = 1 + \sum_{i=1}^{(n-1)^2N/2} 2^i \quad .$$

This clearly would take an enormous amount of memory. Thus it was

decided to work with only one branch at a time. Thus only $(N - 1)^2N/2$ nodes maximum at a time were stored.

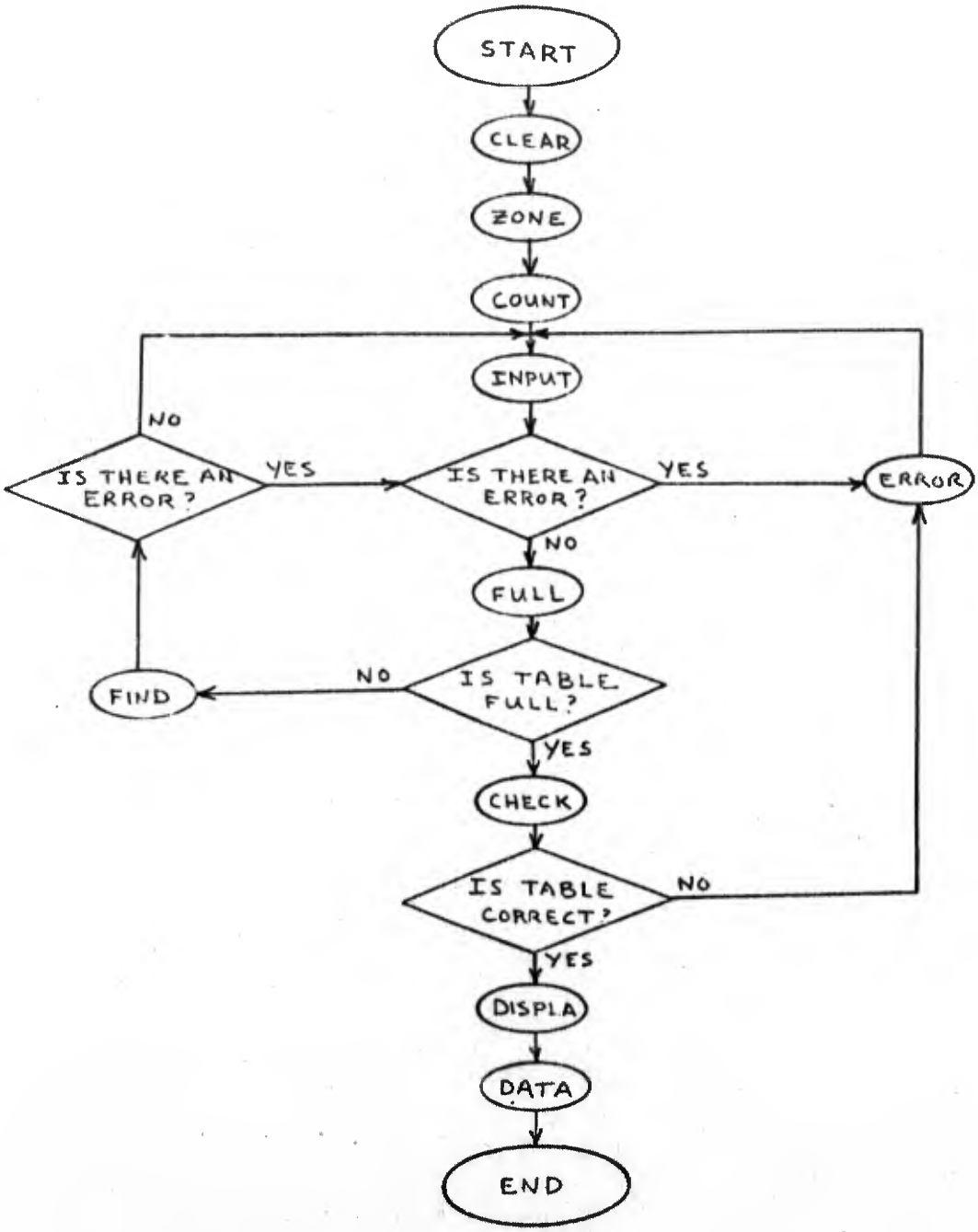
Appendix I also showed an example of the solution of a ten state machine. Its solution required 546 octal inputs versus 54 octal inputs for the solution to Example I of Section 3.3, which contained three states. Thus it is clear that the complexity of the solution as a function of the number of states in the machine is definitely not linear.

The problem of machine identification is well suited for the manufacturing environment. Integrated circuits could be tested as an alternative to fault-detection. Further investigation into the machine identification problem should possibly include the idea of fault-location. If a machine is found to have a fault, then possibly a state table for the fault-containing machine could be derived and compared somehow to the fault-free state table. This might lead to location of the fault.

The algorithm of this paper may also be used to identify machines with multiple inputs and/or multiple outputs. This may be done by re-defining the input and/or output sets to be single variables with more elements. For example, suppose a machine's input sets were $X_1 = \{0, 1\}$ and $X_2 = \{0, 1\}$. The input set could be redefined to be $\overline{X} = \{0,1,2,3\}$. This set could be accepted by the algorithm. Likewise, the output sets could be redefined also.

APPENDIX I

Appendix I contains a listing of a computer program which implements the algorithm described in this paper. A general description of the various routines was given in Chapter IV. This listing also contains comments which describe in detail the functions performed.


START
CLEAR
ZONE
COUNT
INPUT
IS THERE AN ERROR?
NO
YES
IS THERE AN ERROR?
YES
ERROR
NO
FULL
IS TABLE FULL?
NO
FIND
YES
CHECK
IS TABLE CORRECT?
NO
YES
DISPLA
DATA
END

```
          .GLOBL   FSYS
START     DZM      INCL1
          DZM      PIOF
          DZM      ICT
          DZM      PCT
          DZM      INC
          DZM      DISFL
KSF=700301
TSS=700406
TSF=700401
KRB=700312
TCF=700402
          LAC      (IOAD     /IOADD = ADDRESS OF TEST SEQUENCE
          DAC      IOADD
          LAC      (TABLE
          DAC      NS0       /NS0=NEXT ZERO STATE
          TAD      (4
          DAC      NS1       /NS1=NEXT ONE STATE
          JMS      CLEAR
          JMS      ZONE
          LAC      LREP1     /IS LENGTH OF REPEATING X0
          CMA                /EQUAL TO OR GREATER THAN
          TAD      (1        /LENGTH OF REPEATING X1 ?
          TAD      LREP0
          SPA
          SKP                /NO
          ISZ      INCL1     /YES
          JMS      COUNT
          LAC      INCL1
          SZA                /IS ZERO SEQUENCE START OF TEST SEQUENCE ?
          JMP      ST0       /YES
          LAC      S1        /NO
          DAC      SLOC
          DAC      PIO       /PIO = START OF TEST SEQUENCE
          LAC      LREP1
ST1       CMA
          TAD      (3
          DAC      P         /INITIALIZE P
          LAC*     PIO
          AND      (1
          SZA                /IS INITIAL STATE OUTPUT "0" ?
          JMP      ST2       /NO
          LAC      NS0       /YES
          TAD      (10
          DAC      NS0       /UPDATE NS0
          JMP      ST3
ST2       LAC      NS1
          TAD      (10
          DAC      NS1       /UPDATE NS1
ST3       LAC*     PIO
          AND      (1        /GET FIRST OUTPUT
          RCL
          RCL
          TAD      (TABLE
          DAC      PS        /INITIALIZE PRESENT STATE OF M2
          ISZ      PIO       /PIO = NEXT INPUT/OUTPUT COMBINATION
ST4       JMS      INPUT
```

```
        LAC     ERRFL
        SZA                     /IS THERE AN ERROR ?
        SKP                     /YES
        JMP     ST6             /NO
ST5     JMS     ERROR           /CORRECT ERROR
        JMP     ST4
ST6     JMS     FULL            /CHECK TO SEE IF TABLE IS FULL
        LAC     FULF
        SZA                     /IS THE TABLE FULL ?
        JMP     ST7             /YES
        JMS     FIND            /NO
        LAC     ERRFL
        SZA                     /IS THERE AN ERROR ?
        JMP     ST5             /YES
        JMP     ST4             /NO
ST7     JMS     CHECK           /CHECK TABLE FOR CORRECTNESS
        LAC     CKFLA
        SZA                     /IS THE TABLE CORRECT ?
        SKP                     /YES
        JMP     ST5             /NO
        JMS     DISPLA          /PRINT THE CORRECT TABLE
        JMS     DATA            /PRINT PERTINENT DATA
        JMS     END             /PRINT "END"
        HLT
ST8     LAC     S0
        DAC     SLOC            /INITIALIZE TEST SEQUENCE'S START LOCATION
        DAC     PIO             /PIO = START OF TEST SEQUENCE
        LAC     LHEP0           /LOAD L
        JMP     ST1
CLEAR
        LAC     (TABLE
        DAC     TMP1            /TMP1 = ADDRESS OF STATE TABLE
        LAC     (-120
        DAC     CT              /MAXIMUM SIZE OF TABLE IS 80 WORDS
CL1     DZM*    TMP1            /CLEAR TABLE ENTRY
        ISZ     TMP1            /GO TO NEXT WORD
        ISZ     CT              /IS STORAGE AREA CLEAR ?
        JMP     CL1             /NO
        JMP*    CLEAR           /YES
APPLY
        DAC     TMP2            /STORE INPUT TEMPORARILY
        JMS     CRL
        LAC     TMP2            /GET INPUT
        RCL
        DAC*    IOADD           /STORE INPUT IN TEST SEQUENCE
        RCR
        TAD     (260            /PRINT INPUT
        JMS     PRINT
AP1     JMS     READ            /READ CHARACTER
        TAD     (-60
        SZA                     /IS IT A "0" OUTPUT ?
        SKP                     /NO
        JMP     AP4             /YES
        TAD     (-40
        SZA                     /IS IT A DISPLAY REQUEST (P) ?
        JMP     AP2             /NO
        ISZ     DISFL           /YES; SET DISPLAY FLAG
```

```
        JMP     AP1
AP2     TAD     (-141
        SZA                     /IS IT A "1" OUTPUT ?
        JMP     AP3             /NO
        LAC     (1              /YES
        JMP     AP4
AP3     LAC     (277            /PRINT A "?"
        JMS     PRINT
        JMP     AP1
AP4     TAD*    IOADD           /GET INPUT
        DAC*    IOADD           /STORE INPUT/OUTPUT COMBINATION
        ISZ     IOADD
        ISZ     ICT             /UPDATE INPUT COUNT
        JMP*    APPLY
ZONE
        DZM     SEQ             /SEQUENCE = "0"
        LAC     IOADD
        DAC     S0LOC           /S0LOC = START OF ALL ZERO SEQUENCE
Z01     LAC     SEQ
        SZA                     /IS SEQUENCE = "0" ?
        JMP     Z03             /NO
Z02     LAC     (0              /YES
        SKP
Z03     LAC     (1
        JMS     APPLY           /APPLY INPUT
        LAC     ICT
        TAD     (-3
        SPA                     /HAVE AT LEAST 3 INPUTS BEEN APPLIED ?
        JMP     Z02             /NO
        JMS     SERCH           /YES
        LAC     REP             /ARE THERE 3 CYCLES OF A REPEATING OUTPUT
        SZA                     /SEQUENCE ?
        JMP     Z01             /NO
        LAC     SEQ             /YES
        SZA                     /HAVE BOTH ZERO AND ONE SEQUENCES BEEN APPLIED ?
        JMP     Z04             /YES
        LAC     L               /NO
        DAC     LREP0           /LREP0 = LENGTH OF 1 CYCLE OF ALL "0" INPUT SEQUENCE
        LAC     (1
        DAC     SEQ             /INDICATE THAT ALL "1" INPUT SEQUENCE HAS BEGUN
        LAC     IOADD
        DAC     S1LOC           /S1LOC= START OF ALL "1" SEQUENCE
        LAC     X1
        DAC     S0              /S0 = START OF REPEATING OUTPUT
        LAC     Y1              /SEQUENCE FOR "0" INPUTS
        DAC     ES0             /ES0 = END OF REPEATING OUTPUT
        JMP     Z01             /SEQUENCE FOR "0" INPUTS
Z04     LAC     L               /LREP1 = LENGTH OF ONE CYCLE OF
        DAC     LREP1           /ALL "1" INPUT SEQUENCE
        LAC     X1              /S1 = START OF REPEATING OUTPUT
        DAC     S1              /SEQUENCE FOR "1" INPUTS
        LAC     Y1              /ES1 = END OF REPEATING OUTPUT
        DAC     ES1             /SEQUENCE FOR "1" INPUTS
        JMP*    ZONE
SERCH
        LAC     (-2
        DAC     REP             / 3 REPETITIONS OF OUTPUT SEQUENCE ARE DESIRED
```

```
        LAC     SEQ
        SZA                     /IS INPUT SEQUENCE "0" ?
        JMP     SE1             /NO
        LAC     S0LOC           /YES
        DAC     XX              /XX = ADDRESS OF "0" SEQUENCE
        JMP     SE2
SE1     LAC     S1LOC
        DAC     XX              /XX = ADDRESS OF "1" SEQUENCE
SE2     LAC     XX
        TAD     (1
        DAC     Y               /Y = XX + 1
        JMP     SE5
SE3     JMS     EQUAL
        LAC     EQ
        SZA                     /IS OUTPUT AT XX = OUTPUT AT Y ?
        JMP     SE6             /YES
SE4     ISZ     Y               /NO
SE5     LAC     Y
        CMA
        TAD     IOADD
        SPA                     /IS Y AT END OF TEST SEQUENCE ?
        SKP                     /YES
        JMP     SE3             /NO
        ISZ     XX              /GO TO NEXT OUTPUT AND REPEAT CHECK
        LAC     XX
        CMA
        TAD     IOADD
        SPA                     /IS XX AT END OF TEST SEQUENCE ?
        JMP*    SERCH           /YES
        JMP     SE2             /NO
SE6     LAC     Y
        DAC     Y1              /Y1 = END OF A REPEATING OUTPUT SEQUENCE
        LAC     XX
        DAC     X1              /X1 = START OF A REPEATING OUTPUT SEQUENCE
        CMA
        TAD     (1
        TAD     Y1
        DAC     L               /L = Y1-X1= LENGTH OF REPEATING SEQUENCE
        LAC     Y1
        TAD     (-1             /UB IS A TEMPORARY UPPER BOUND TO DETERMINE
        DAC     UB              /WHEN A COMPLETE CYCLE HAS BEEN FOUND
SE7     ISZ     XX              /UPDATE XX AND Y TO CONTINUE COMPARISON
        ISZ     Y               /OF OUTPUTS
        LAC     Y
        CMA
        TAD     IOADD
        SPA                     /IS Y AT END OF TEST SEQUENCE ?
        JMP*    SERCH           /YES
        JMS     EQUAL           /NO
        LAC     EQ
        SZA                     /IS (OUTPUT AT Y= OUTPUT AT XX) ?
        JMP     SE8             /YES
        LAC     X1              /NO) CHECK NEXT OUTPUT FOR REPEAT
        DAC     XX
        LAC     Y1
        DAC     Y
        LAC     (-2
```

```
        DAC     REP     /REINITIALIZE REP
        JMP     SE4
SE8     LAC     UB
        CMA
        TAD     (1
        TAD     XX      /HAS ONE CYCLE OF REPEATING
        SPA             /OUTPUT SEQUENCE BEEN CHECKED ?
        JMP     SE7     /NO
        LAC     UB      /YES; CHECK NEXT CYCLE
        TAD     L
        DAC     UB      /MOVE UB TO NEXT BOUNDARY
        ISZ     REP     /HAVE 3 CYCLES BEEN FOUND ?
        JMP     SE7     /NO
        JMP*    SERCH   /YES
EQUAL
        DZM     EQ      /THIS FLAG INDICATES 2 OUTPUTS ARE EQUAL
        LAC*    XX
        AND     (1      /GET FIRST OUTPUT
        DAC     TMP1
        LAC*    Y
        AND     (1      /GET NEXT OUTPUT TO BE COMPARED
        TAD     TMP1
        SZA             /ARE OUTPUTS EQUAL ?
        JMP     EQ1     /MAYBE
        JMP     EQ2     /YES
EQ1     TAD     (-2
        SZA             /ARE OUTPUTS EQUAL ?
        JMP     EQ3     /NO
EQ2     ISZ     EQ      /YES; SET FLAG
EQ3     JMP*    EQUAL
COUNT
        DZM     TMP3    /TMP3 = # OF ZEROS
        DZM     TMP4    /TMP4 = # OF ONES
        LAC     INCL1
        SZA             /IS START OF TEST SEQUENCE S1 ?
        JMP     CO4     /NO
        LAC     S1      /YES
        DAC     TMP1
        LAC     ES1
        DAC     TMP2
CO1     LAC*    TMP1    /GET AN OUTPUT
        AND     (1
        SZA             /IS OUTPUT "0" ?
        JMP     CO5     /NO
        ISZ     TMP3    /YES; INCREMENT "0" COUNT
CO2     ISZ     TMP1    /GO TO NEXT OUTPUT
        LAC     TMP1
        SAD     TMP2    /HAS COMPLETE SEQUENCE BEEN COUNTED ?
        SKP             /YES
        JMP     CO1     /NO
        LAC     TMP4
        CMA
        TAD     (1
        TAD     TMP3    /ACCUMULATOR CONTAINS TMP3-TMP4
        SPA             /IS # OF "0" OUTPUTS >= # OF "1" OUTPUTS ?
        JMP     CO6     /NO
        LAC     TMP3    /YES
```

```
C03     RCL
        DAC     N           /INITIALIZE N
        RCL
        RCL
        TAD     (TABLE
        DAC     LTAB        /LTAB = ADDRESS OF STATE TABLE'S END
        JMP*    COUNT
C04     LAC     S0
        DAC     TMP1
        LAC     ES0
        DAC     TMP2
        JMP     C01
C05     ISZ     TMP4        /INCREMENT "1" COUNT
        JMP     C02
C06     LAC     TMP4
        JMP     C03
INPUT
IN1     LAC     DISFL
        SZA                 /WAS THERE A DISPLAY REQUEST ?
        JMS     DISPLA      /YES
        DZM     DISFL       /NO
        KSF                 /IS THERE A READ IN REQUEST ?
        JMP     IN3         /NO
        JMS     READ        /YES
        TAD     (-120
        SZA                 /IS IT A DISPLAY REQUEST (P) ?
        JMP     IN2         /NO
        JMS     DISPLA      /YES
        JMP     IN3
IN2     LAC     (277
        JMS     PRINT       /PRINT A "?"
IN3     LAC     IOADD
        CMA
        TAD     (1
        TAD     PIO
        SPA                 /IS PIO = TO END OF TEST SEQUENCE ?
        SKP                 /NO
        JMP*    INPUT       /YES
        LAC*    PIO         /GET NEXT OUTPUT
        AND     (1
        DAC     POUT        /POUT IS PRESENT OUTPUT
        LAC*    PIO
        AND     (2
        DAC     PINP        /PINP IS PRESENT INPUT
        SZA                 /IS PINP A "0" ?
        JMP     IN7         /NO
        LAC     PS          /YES
IN4     DAC     TMP5        /TMP5 = ADDRESS WHERE NEXT STATE ENTRY GOES
        LAC*    TMP5
        SZA                 /IS PRESENT STATE ENTRY BLANK ?
        JMP     IN8         /NO
        LAC     P           /YES
        DAC     TMP3
        SMA                 /IS P > 0 ?
        SKP                 /MAYBE
        JMP     IN9         /NO
        SZA                 /IS P > 0 ?
```

```
        SKP             /YES
        JMP     IN9     /NO
        LAC*    PIO     /GENERATE NEXT STATE ENTRY
        AND     (1
        RCL
        RCL
        TAD     (TABLE  /NS = STATE NUMBER (1 + 4 * OUTPUT)
IN5     DAC*    TMP5    /ENTER NEXT STATE IN TABLE
        ISZ     TMP5    /INCREMENT TMP5 TO STORE P
        LAC     P
        DAC*    TMP5    /ENTER P IN TABLE
        LAC     TMP5
        TAD     (-1
        DAC     TMP5    /MOVE TMP5 BACK TO NS ENTRY
        LAC     P
        RCL
        TAD     (ENTAB
        DAC     TMP2    /ADDRESS IN TABLE OF ENTRIES
        LAC     PS
        DAC*    TMP2    /STORE PS IN TABLE OF ENTRIES
        ISZ     TMP2
        LAC     PIO
        DAC*    TMP2    /STORE PIO IN TABLE OF ENTRIES
IN6     LAC     TMP3
        TAD     (1
        DAC     P       /UPDATE P
        LAC*    TMP5
        DAC     PS      /UPDATE PRESENT STATE
        ISZ     PIO     /MOVE TO NEXT INPUT/OUTPUT COMBINATION
        JMP     IN1
IN7     LAC     PS
        TAD     (2      /INPUT IS "1"
        JMP     IN4
IN8     LAC     (TABLE
        CMA
        TAD     (1
        TAD*    TMP5    /GET OUTPUT INDICATED BY M2
        RCR
        RCR
        AND     (1
        SAD     POUT    /IS POUT = OUTPUT INDICATED BY M2 ?
        JMP     IN12    /YES
        LAC     (1      /NO; SET ERROR FLAG
        DAC     ERRFL
        JMP*    INPUT
IN9     DZM     P
        LAC     POUT
        SZA             /IS M1 OUTPUT A "0" ?
        JMP     IN11    /NO
        LAC     NS0     /YES
        DAC     TMP1    /TMP1 CONTAINS THE NEXT STATE OF M2
        TAD     (10
        DAC     NS0     /UPDATE NS0
IN10    LAC     TMP1    /TMP1 CONTAINS NEXT STATE TO BE ENTERED IN TABLE
        JMP     IN5
IN11    LAC     NS1
        DAC     TMP1    /TMP1 CONTAINS THE NEXT STATE OF M2
```

```
        TAD     (10
        DAC     NS1         /UPDATE NS1
        JMP     IN10
IN12    ISZ     PIO         /PIO = PIO + 1
        LAC*    TMP5
        DAC     PS          /UPDATE PS
        JMP     IN1
FIND
        LAC*    PS
        SZA                 /DOES THE PRESENT STATE HAVE A BLANK ENTRY ?
        JMP     FI2         /MAYBE
        CLA                 /YES; PINP IS "0"
FI1     JMS     APPLY       /APPLY THE INPUT
        JMP*    FIND
FI2     LAC     PS
        TAD     (2          /INPUT IS A "1"
        DAC     PS
        LAC*    PS
        SZA                 /DOES THE PRESENT STATE HAVE A BLANK ENTRY ?
        JMP     FI3         /NO
        LAC     PS          /YES
        TAD     (-2
        DAC     PS
        LAC     (1          /PINP IS "1"
        JMP     FI1
FI3     LAC     PS
        TAD     (-2
        DAC     PS
        LAC     (TABLE
        DAC     PT4         /PT4 IS A POINTER USED TO LOCATE
FI4     LAC*    PT4         /A STATE WHICH HAS ONLY ONE BLANK ENTRY.
        SZA                 /IS THERE A BLANK ENTRY ?
        SKP                 /NO
        JMP     FI6         /YES
        LAC     PT4
        TAD     (2
        DAC     PT4         /MOVE TO NEXT ENTRY
        LAC*    PT4
        SZA                 /IS "1" INPUT ENTRY BLANK ?
        SKP                 /NO
        JMP     FI7         /YES
FI5     LAC     PT4
        TAD     (2
        DAC     PT4         /MOVE TO NEXT ENTRY
        JMP     FI4
FI6     LAC     PT4
        TAD     (2
        DAC     PT4         /MOVE TO NEXT ENTRY
        LAC*    PT4
        SZA                 /IS IT THE ONLY BLANK ENTRY ?
        JMP     FI7         /YES
        JMP     FI5         /NO; CHECK NEXT ROW
FI7     LAC     PT4
        TAD     (-2         /THERE IS ONLY ONE BLANK ENTRY IN THE STATE
        DAC     FPS         /FPS=FUTURE PRESENT STATE, FPS IS THE STATE
        SKP                 /TO WHICH M2 IS TO TRANSITION.
SSFIND
```

```
        LAC     N
        RCL
        TAD     (-1
        DAC     MSEQ        /2N-1=MSEQ=MAXIMUM LENGTH OF SEQUENCE
        DZM     FSEQ        /FSEQ IS A FLAG WHICH INDICATES IF A SEQUENCE EXISTS
        DZM     CT4
        LAC     (FITAB      /FITAB IS ADDRESS WHERE NEXT STATE
        DAC     PT4         /ENTRIES ARE KEPT
        LAC     PS
        DAC*    PT4         /PT4 IS AN ADDRESS IN FITAB
        LAC     PS
        SAD     FPS         /IS PS = FPS ?
        JMP*    SSFIND      /YES
        DZM     FINP        /NO; FINP IS NEXT INPUT IN THE SEQUENCE
FI8     LAC     PT4
        TAD     (1
        DAC     PT5         /PT5 CONTAINS ADDRESS WHERE INPUT IS STORED
        ISZ     CT4         /CT4 CONTAINS LENGTH OF SEQUENCE
        LAC     FINP        /GET NEXT INPUT
        DAC*    PT5         /STORE INPUT
        ISZ     PT5
        LAC*    PT4
        TAD     FINP
        DAC     PPS         /PPS CONTAINS NEXT STATE IN LINE TO FPS
        LAC*    PPS         /GET NEXT STATE
        DAC*    PT5         /STORE NEXT STATE IN FITAB
        SAD     FPS         /HAS SEQUENCE REACHED FPS ?
        JMP     FI15        /YES
        DZM     FINP        /NO
        LAC     MSEQ
        CMA
        TAD     (1
        TAD     CT4
        SPA                 /HAS SEQUENCE LENGTH REACHED MAXIMUM ?
        SKP                 /NO; ADD ANOTHER INPUT TO SEQUENCE
        JMP     FI9         /YES
        ISZ     PT4
        ISZ     PT4
        JMP     FI8
FI9     LAC     PT5
        TAD     (-1
        DAC     PT5
        CMA
        TAD     (1
        TAD     (FITAB
        SPA                 /HAVE ALL SEQUENCES BEEN TRIED ?
        SKP                 /NO
        JMP     FI11        /YES
        LAC*    PT5         /GET LAST INPUT
        SZA                 /WAS LAST INPUT A "0" ?
        JMP     FI10        /NO
        LAC     (2          /YES
        DAC     FINP        /FINP = 1
        LAC     CT4
        TAD     (-1
        DAC     CT4         /CORRECT INPUT COUNT
        JMP     FI8
```

```
FI10    LAC     CT4
        TAD     (-1
        DAC     CT4         /CORRECT INPUT COUNT
        LAC     PT5
        TAD     (-1
        DAC     PT5         /CORRECT INPUT STORAGE ADDRESS
        LAC     PT4
        TAD     (-2
        DAC     PT4         /CORRECT NEXT STATE STORAGE ADDRESS
        JMP     FI9
FI11    LAC     FSEQ
        SZA                 /IS THERE A SEQUENCE ?
        JMP     FI12        /YES; APPLY THE SEQUENCE TO M1
        LAC     (1          /NO
        DAC     ERRFL       /SET ERROR FLAG
        JMP     FI14
FI12    LAC     PT8
        TAD     (-1
        DAC     PT8         /MOVE TO NEXT INPUT
FI13    LAC*    PT8         /LOAD AN INPUT
        RCR
        JMS     APPLY       /APPLY THE INPUT TO M1
        JMS     INPUT       /APPLY THE INPUT TO M2
        LAC     ERRFL
        SZA                 /DOES M2 INDICATE AN ERROR ?
        JMP     FI14        /YES
        LAC     PT8         /NO
        TAD     (-1
        DAC     PT8         /MOVE TO NEXT INPUT
        LAC     (FITAB2
        CMA
        TAD     (1
        TAD     PT8
        SPA                 /HAS THE FULL SEQUENCE BEEN APPLIED ?
        SKP                 /YES
        JMP     FI13        /NO
FI14    LAC     CKFLA
        SZA                 /WAS ROUTINE CALLED BY "CHECK" ROUTINE ?
        JMP*    SSFIND      /YES
        JMP*    FIND        /NO
FI15    LAC     PT5
        TAD     (-1
        DAC     PT7         /PT7 IS POINTING AT LAST INPUT
        LAC     (FITAB2     /FITAB2 = LOCATION OF SEQUENCE
        DAC     PT8         /PT8 IS A POINTER FOR STORING SEQUENCE
        LAC     (FITAB
        CMA
        TAD     (1
        TAD     PT5
        RCR
        DAC     SEQL        /SEQL = LENGTH OF SEQUENCE
        TAD     (-1
        DAC     MSEQ        /UPDATE MSEQ AND CONTINUE SEARCH
        LAC     (1          /SET FLAG TO INDICATE THAT A
        DAC     FSEQ        /SEQUENCE HAS BEEN FOUND
        LAC     SEQL
        CMA
```

```
        TAD     (1
        DAC     CT3         /CT3 = -SEQL
FI16    LAC*    PT7         /STORE THE INPUT
        DAC*    PT8         /STONE IN FITAB2
        LAC     PT7
        TAD     (-2
        DAC     PT7         /GET NEXT INPUT
        ISZ     PT8         /UPDATE FITAB2 STORAGE POINTER
        ISZ     CT3         /HAS ALL INPUTS BEEN STORED ?
        JMP     FI16        /NO
        LAC     PT4         /YES
        DAC     PT5         /CORRECT INPUT STORAGE POINTERS
        TAD     (-2         /AND CONTINUE LOOKING FOR A
        DAC     PT4         /SHORTER SEQUENCE
        LAC     CT4
        TAD     (-1
        DAC     CT4
        JMP     FI9
ERROR
        DZM     ERRFL       /RESET ERROR FLAG
ER1     LAC     P
        TAD     (-1
        DAC     P           /CORRECT P
        SZA                 /IS TABLE TO SMALL ?
        JMP     ER2         /NO
        ISZ     N           /YES; MODIFY M2 AND TABLE OF ENTRIES
        ISZ     N
        LAC     LTAB
        TAD     (13
        DAC     LTAB        /CORRECT LTAB
        LAC     ENTAB       /GET INITIAL PS
        DAC     TMP1
        LAC*    ENTAB+1     /GET INITIAL INPUT
        AND     (2
        TAD     TMP1
        DAC     TMP1
        LAC*    TMP1        /GET SI(1) FROM TABLE
        DAC     PS          /INITIALIZE PRESENT STATE
        LAC     ENTAB+1
        DAC     PIO         /INITIALIZE PIO
        ISZ     PIO
        ISZ     P           / P = 1
        JMP*    ERROR
ER2     ISZ     INC         /INCREMENT INCORRECT ENTRY COUNT
        LAC     P
        RCL
        TAD     (ENTAB
        DAC     TMP1        /TMP1 CONTAINS ADDRESS OF LAST ENTRY IN M2
        ISZ     TMP1
        LAC*    TMP1        /GET PIO WHERE LAST ENTRY WAS MADE
        DAC     PIO
        LAC     TMP1
        TAD     (-1
        DAC     TMP1
        LAC*    TMP1        /GET STATE WHERE LAST ENTRY WAS MADE
        DAC     TMP5        /STORE TEMPORARILY
        LAC*    PIO
```

```
        AND     (2
        DAC     PINP        /GET PROPER INPUT
        TAD     TMP5
        DAC     TMP5
        LAC*    TMP5        /GET LAST ENTRY
        DAC     TMP6
        LAC*    TMP6
        SZA                 /IS NEXT STATE AN EMPTY ROW ?
        JMP     ER3         /NO
        ISZ     TMP6        /MAYBE
        ISZ     TMP6
        LAC*    TMP6
        SZA                 /IS NEXT STATE AN EMPTY ROW ?
        SKP                 /NO
        JMP     ER4         /ES
ER3     LAC*    TMP5
        TAD     (10         /CHANGE SI TO SI*2
        DAC*    TMP5
        LAC     LTAB
        CMA
        TAD     (1
        TAD*    TMP5
        SPA                 /IS SI*2 OUTSIDE THE TABLE ?
        JMP     ER5         /NO
ER4     DZM*    TMP5        /YES: CLEAR M2 ENTRY
        ISZ     TMP5
        DZM*    TMP5        /CLEAR SECOND WORD OF ENTRY
        JMP     ER1
ER5     LAC*    TMP5
        DAC     PS          /UPDATE PS
        ISZ     PIO         /UPDATE PIO
        ISZ     P           /UPDATE P
        JMP*    ERROR
FULL
        DZM     FULF
        DZM     INP
        LAC*    PS
        SNA                 /DOES PS HAVE A BLANK ENTRY ?
        JMP     FU3         /YES
        LAC     N           /NO
        CMA
        TAD     (1
        RCL
        RCL
        DAC     TBE         /TBE =  -ROW COUNT
        LAC     (TABLE
        DAC     TBB         /TBB = TABLE BEGINNING
FU1     LAC*    TBB
        SZA                 /IS ENTRY BLANK ?
        JMP     FU4         /NO
        LAC     INP         /YES
        AND     (1
        SZA                 /IS BLANK ENTRY IN "0" INPUT COLUMN ?
        JMP     FU3         /NO
        LAC     TBB         /YES
        TAD     (2
        DAC     TMP1
```

```
        LAC*    TMP1
        SZA             /ARE BOTH ENTRIES BLANK ?
        JMP     FU3     /NO
        LAC     TBB     /YES; CHECK NEXT ROW
        TAD     (4
        DAC     TBB
        ISZ     TBE
FU2     ISZ     TBE     /HAVE ALL ROWS BEEN CHECKED ?
        JMP     FU1     /NO
        LAC     (1      /YES; SET FULL FLAG
        DAC     FULF
FU3     JMP*    FULL
FU4     LAC     TBB
        TAD     (2
        DAC     TBB     /MOVE TO NEXT ENTRY
        ISZ     INP     /CHANGE INPUT COLUMNS
        JMP     FU2
CHECK
        JMS     DISPLA
        LAC     (1
        DAC     CKFLA   /INDICATE CHECK ROUTINE IS IN OPERATION
        DZM     FULF    /RESET FULL FLAG
        JMS*    FSYS    /FIND SYNCHRONIZING SEQUENCE
        JMP     .+7
        TABLE
        SYNFL
        SYNT2
        N
        SYWORC
        SYBITC
CH1     LAC     SYNFL
        SNA             /IS THERE A SYNCHRONIZING SEQUENCE ?
        JMP     CH4     /NO
        LAC     (1      /YES
        DAC     SYINP   /SYINP = THE PRESENT INPUT TO SEQUENCE
        LAC     N
        CMA
        TAD     (1
        DAC     CT1     /CT1 IS A STATE COUNT
        LAC     (TABLE
        DAC     PT9     /PT9 IS AN ADDRESS IN THE STATE TABLE
CH2     ISZ     SYINP
        JMS     SYSAPP  /APPLY SYNCHRONIZING SEQUENCE
        LAC     ERRFL
        SZA             /IS M2 IN ERROR ?
        JMP     CH4     /YES
        LAC     PT9     /NO
        DAC     FPS     /FPS = STATE TO BE CHECKED
        JMS     SSFIND  /FIND A SEQUENCE TO STATE TO BE CHECKED
        LAC     ERRFL
        SZA             /IS THERE AN ERROR IN M2 ?
        JMP     CH4     /YES
        LAC     SYINP   /NO
        AND     (1      /GET PROPER INPUT
        JMS     APPLY   /APPLY CHECK INPUT TO M1
        JMS     INPUT   /APPLY CHECK INPUT TO M2
        LAC     ERRFL
```

```
        SZA             /IS THERE AN ERROR ?
        JMP     CH4     /YES
        LAC     SYINP   /NO
        AND     C1      /GET LAST INPUT
        SZA             /HAVE BOTH INPUTS BEEN CHECKED ?
        SKP             /YES
        JMP     CH2     /NO
CH3     LAC     PT9
        TAD     C4
        DAC     PT9     /MOVE TO NEXT STATE
        LAC     N
        RCL
        RCL
        TAD     CTABLE
        CMA
        TAD     C1
        TAD     PT9
        SPA             /HAVE ALL STATES BEEN CHECKED ?
        SKP             /NO
        JMP*    CHECK   /YES
        LAC*    PT9
        SZA             /IS NEXT ROW BLANK ?
        JMP     CH2     /NO
        JMP     CH3     /YES; GO TO NEXT ROW
CH4     DZM     CKFLA   /AN ERROR WAS FOUND
        JMP*    CHECK
SYSAPP
        LAC     SYWORC  /SYWORC = # OF WORDS CONTAINING INPUTS
        CMA             /EACH WORD CONTAINS 18 INPUTS
        TAD     C1
        DAC     CT2     /CT2 CONTAINS NUMBER OF WORDS CONTAINING INPUTS
        LAC     SYBITC  /SYBITC = # OF INPUTS IN LAST WORD OF INPUTS
        CMA
        TAD     C1
        DAC     CT1     /CT1 CONTAINS NUMBER OF INPUTS IN A WORD
        LAC     CSYNT2  /SYNT2 = LOCATION OF STORED SYNCHRONIZING SEQUENCE
        TAD     SYWORC
        TAD     C-1
        DAC     SYWO    /SYWO IS THE ADDRESS OF A WORD OF INPUTS
        LAC*    SYWO    /GET FIRST WORD OF INPUTS
        DAC     TMP6
SY1     LAC     TMP6    /TMP6 CONTAINS A WORD OF INPUTS
        RCL             /SHIFT NEXT INPUT INTO LINK
        DAC     TMP6
        SZL             /IS INPUT A "0" ?
        JMP     SY2     /NO
        CLA             /YES
        SKP
SY2     LAC     C1
        JMS     APPLY   /APPLY INPUT TO M1
        JMS     INPUT   /APPLY INPUT TO M2
        LAC     ERRFL
        SZA             /IS THERE AN ERROR ?
        JMP*    SYSAPP  /YES
        ISZ     CT1     /NO
        JMP     SY1
        ISZ     CT2     /HAS COMPLETE SEQUENCE BEEN APPLIED ?
```

```
        SKP             /NO
        JMP*    SYSAPP  /YES
        LAC     (-22
        DAC     CT1     /CORRECT CT1 AND CONTINUE APPLYING SEQUENCE
        LAC     SYWO
        TAD     (-1
        DAC     SYWO    /MOVE TO NEXT WORD OF INPUTS
        LAC*    SYWO    /GET NEXT WORD OF INPUTS
        DAC     TMP6
        JMP     SY1
DISPLA
        LAC     (1
        DAC     CTP     /CTP CONTAINS THE STATE NUMBER
        LAC     (TABLE
        DAC     TMP2    /TMP2 POINTS TO ROW TO BE PRINTED
        CMA
        TAD     (1
        DAC     TMP3    /TMP3 = -(ADDRESS OF THE TABLE)
        LAC     (-4     /PCT CONTAINS THE NUMBER OF TIMES A
        DAC     PCT     /CHARACTER IS TO BE REPEATED
        JMS     CRL
        LAC     (320
        JMS     PRINT   /PRINT A "P"
        LAC     (323
        JMS     PRINT   /PRINT AN "S"
        LAC     (-14
        DAC     PCT
        JMS     BLNKS
        LAC     (316
        JMS     PRINT   /PRINT AN "N"
        LAC     (323
        JMS     PRINT   /PRINT AN  "S"
        LAC     (-12
        DAC     PCT
        JMS     BLNKS
        LAC     (332
        JMS     PRINT   /PRINT A "Z"
        JMS     CRL
        LAC     (-11
        DAC     PCT
        JMS     BLNKS
        LAC     (260
        JMS     PRINT   /PRINT A "0"
        LAC     (-11
        DAC     PCT
        JMS     BLNKS
        LAC     (261
        JMS     PRINT   /PRINT A "1"
        JMS     CRL
DI1     LAC     CKFLA
        SNA             /IS FINAL TABLE BEING PRINTED ?
        JMP     DI2     /NO
        LAC*    TMP2    /YES
        SZA             /IS "0" ENTRY BLANK ?
        JMP     DI2     /NO
        LAC     TMP2    /YES
        TAD     (2
```

```
        DAC     TMP1
        LAC*    TMP1
        SZA                     /IS "1" ENTRY BLANK ?
        JMP     DI2             /NO
        ISZ     CTP             /YES  INCREMENT STATE NUMBER
        LAC     TMP2
        TAD     (4
        DAC     TMP2            /MOVE TO NEXT ROW
        LAC     N
        RCL
        RCL
        TAD     (TABLE
        CMA
        TAD     (1
        TAD     TMP2
        SPA                     /HAS  COMPLETE TABLE BEEN PRINTED ?
        JMP     DI1             /NO
        JMP     DI9             /YES
DI2     LAC     (323
        JMS     PRINT           /PRINT AN "S"
        LAC     CTP
        RCR
        RCR
        RCR
        SZA                     /IS STATE NUMBER < 10 ?
        JMP     DI3             /NO
        JMS     BLNKS           /YES  PRINT A SPACE
        JMP     DI4
DI3     TAD     (260
        JMS     PRINT           /PRINT MOST SIGNIFICANT DIGIT OF STATE NUMBER
DI4     LAC     CTP
        AND     (7
        TAD     (260
        JMS     PRINT           /PRINT LEAST SIGNIFICANT DIGIT OF STATE NUMBER
        ISZ     CTP             /INCREMENT STATE NUMBER
        LAC     (-2
        DAC     CTT             /CTT INDICATES THAT 2 COLUMNS ARE TO BE PRINTED
DI5     LAC     (-3
        DAC     PCT
        JMS     BLNKS
        LAC*    TMP2
        SZA                     /IS ENTRY BLANK ?
        SKP                     /NO
        JMP     DI10            /YES
        LAC     (323
        JMS     PRINT           /PRINT AN "S"
        LAC*    TMP2            /GET ENTRY TO BE PRINTED
        TAD     TMP3
        RCR
        RCR
        TAD     (1
        DAC     TMP4
        RCR
        RCR
        RCR
        SZA                     /IS STATE NUMBER < 10 ?
        SKP                     /NO
```

```
        JMP     D111    /YES
        TAD     (260
        JMS     PRINT   /PRINT MOST SIGNIFICANT DIGIT OF STATE NUMBER
D116    LAC     TMP4
        AND     (7
        TAD     (260
        JMS     PRINT   /PRINT LEAST SIGNIFICANT DIGIT OF STATE NUMBER
        ISZ     TMP2
        LAC*    TMP2    /GET P VALUE
        SZA             /IS P = 0 ?
        SKP             /NO
        JMP     D112    /YES
        LAC     (250
        JMS     PRINT   /PRINT "("
        LAC*    TMP2
        RCR
        RCR
        RCR
        SZA             /IS P < 10 ?
        SKP             /NO
        JMP     D113    /YES
        TAD     (260
        JMS     PRINT   /PRINT MOST SIGNIFICANT DIGIT OF P
D117    LAC*    TMP2
        AND     (7
        TAD     (260
        JMS     PRINT   /PRINT LEAST SIGNIFICANT DIGIT OF P
        ISZ     TMP2
        LAC     (251
        JMS     PRINT   /PRINT A ")"
D118    ISZ     CTT     /HAVE BOTH COLUMNS BEEN PRINTED ?
        JMP     D115    /NO
        LAC     (-3     /YES
        DAC     PCT
        JMS     BLNKS
        LAC     CTP
        AND     (1      /GET STATE'S OUTPUT
        TAD     (260
        JMS     PRINT   /PRINT STATE OUTPUT
        JMS     CRL
        LAC     CTP
        CMA
        TAD     (1
        TAD     N
        SMA             /HAS COMPLETE TABLE BEEN PRINTED ?
        JMP     D11     /NO
D119    LAC     (-4     /YES
        DAC     PCT
        JMS     CRL
        JMP*    DISPLA
D1110   LAC     (-7
        DAC     PCT
        JMS     BLNKS
        ISZ     TMP2
        ISZ     TMP2    /MOVE TO NEXT COLUMN
        JMP     D118
D111    JMS     BLNKS
```

```
          JMP       D16
D112      LAC       (-4
          DAC       PCT
          JMS       BLNKS
          ISZ       TMP2        /MOVE TO NEXT WORD IN TABLE
          JMP       D18
D113      JMS       BLNKS
          JMP       D17
PRINT
          DAC       TMP1
          LAC       PCT
          SZA                   /IS ONE CHARACTER TO BE PRINTED ?
          JMP       PR2         /NO
          LAC       (-1         /YES
          DAC       PCT
PR2       IOF
          CLA
          ISA
PR1       LAC       TMP1
          TLS                   /PRINT
          TSF
          JMP       .-1
          ISZ       PCT         /HAVE ALL CHARACTERS BEEN PRINTED ?
          JMP       PR1         /NO
          TCF                   /YES
          LAC       (400000
          ISA
          ION
          JMP*      PRINT
READ
          IOF
          CLA
          ISA
          KSF
          JMP       .-1
          KRB                   /READ
          DAC       TMP1
          LAC       (400000
          ISA
          ION
          LAC       TMP1
          JMP*      READ
DATA
          LAC       (-3
          DAC       PCT
          JMS       CRL
          JMS       MSG1
          JMS       MSG2
          LAC       (-3
          DAC       PCT
          JMS       BLNKS
          LAC       ICT
          JMS       OCTAL       /PRINT ICT IN OCTAL NOTATION
          JMS       CRL
          JMS       MSG1
          JMS       MSG3
          LAC       (-3
```

```
            DAC     PCT
            JMS     BLNKS
            LAC     INC
            JMS     OCTAL       /PRINT INC IN OCTAL NOTATION
            JMS     CRL
            JMS     MSG1
            JMS     MSG2
            LAC     (257
            JMS     PRINT       /PRINT A "/"
            JMS     MSG1
            JMS     MSG3
            LAC     (275
            JMS     PRINT       /PRINT "="
            LAC     INC
            DAC     DA1
            LAC     ICT
            CLL
            IOIV                /CALCULATE ICT/INC
DA1         0
            DAC     REMAIN
            LACQ
            JMS     OCTAL       /PRINT QUOTIENT IN OCTAL NOTATION
            JMS     BLNKS
            LAC     (253
            JMS     PRINT       /PRINT "+"
            JMS     BLNKS
            LAC     RLMAIN
            JMS     OCTAL       /PRINT REMAINDER IN OCTAL NOTATION
            LAC     (257
            JMS     PRINT       /PRINT A "/"
            LAC     INC
            JMS     OCTAL       /PRINT INC IN OCTAL NOTATION
            JMS     CRL
            JMP*    DATA
MSG1                            /PRINT "# OF "
            LAC     (243
            JMS     PRINT
            LAC     (240
            JMS     PRINT
            LAC     (317
            JMS     PRINT
            LAC     (306
            JMS     PRINT
            LAC     (240
            JMS     PRINT
            JMP*    MSG1
MSG2                            /PRINT "INPUTS"
            LAC     (311
            JMS     PRINT
            LAC     (316
            JMS     PRINT
            LAC     (320
            JMS     PRINT
            LAC     (325
            JMS     PRINT
            LAC     (324
            JMS     PRINT
```

```
          LAC     (323
          JMS     PRINT
          JMP*    MSG2
MSG3                        /PRINT "INCORRECT ENTRIES"
          LAC     (311
          JMS     PRINT
          LAC     (316
          JMS     PRINT
          LAC     (303
          JMS     PRINT
          LAC     (317
          JMS     PRINT
          LAC     (322
          JMS     PRINT
          LAC     (322
          JMS     PRINT
          LAC     (305
          JMS     PRINT
          LAC     (303
          JMS     PRINT
          LAC     (324
          JMS     PRINT
          LAC     (240
          JMS     PRINT
          LAC     (305
          JMS     PRINT
          LAC     (316
          JMS     PRINT
          LAC     (324
          JMS     PRINT
          LAC     (322
          JMS     PRINT
          LAC     (311
          JMS     PRINT
          LAC     (305
          JMS     PRINT
          LAC     (323
          JMS     PRINT
          JMP*    MSG3
BLNKS                       /PRINT SPACES
          LAC     (240
          JMS     PRINT
          JMP*    BLNKS
OCTAL                       /PRINT NUMBER IN OCTAL NOTATION
          LMQ
          LAC     (-6
          DAC     CNT
OC1       CLA
          LLS 3
          TAD     (60
          JMS     PRINT
          ISZ     CNT
          JMP     OC1
          JMP*    OCTAL
END                         /PRINT "END"
          JMS     CRL
          LAC     (305
```

```
                JMS     PRINT
                LAC     (316
                JMS     PRINT
                LAC     (304
                JMS     PRINT
                JMS     CRL
                JMP*    END
CRL                                     /EXECUTE A CARRIAGE RETURN AND LINE FEED
                LAC     (12
                JMS     PRINT
                LAC     (15
                JMS     PRINT
                JMP*    CRL
INCL1
CKFLA
ERRFL
FULF
REP
SEQ
EQ
IFLAG
OKFLA
POUT
PINP
PIOF
NS0
NS1
CT
ICT
BICT
WICT
TBE
TBB
INC
CTP
CTT
ATABLE
CT1
CT2
CT3
CT4
PT4
PT5
PT6
PT7
PT8
PT9
SEOL
MSEQ
FSEQ
FINP
CNT
PPS
FPS
INP
HUNT
USFPS
```

```
LOC
PCT
DISFL
IOADO
PS
PIO
INORO
XX
Y
X1
Y1
S1
S0
ES0
ES1
S0LOC
S1LOC
LREP0
LREP1
SLOC
L
P
N
LTAB
UB
TMP1
TMP2
TMP3
TMP4
TMP5
TMP6
REMAIN
SYINP
SYNFL
SYWORC
SYBITC
SYWO
SYNT2      .BLOCK     210
FPSLOC     .BLOCK     24
FITAB      .BLOCK     40
FITAB2     .BLOCK     40
ENTAB      .BLOCK     50
TABLE      .BLOCK     120
IOAD       .BLOCK     1000
           .END START
```

```
        .GLOBL  .DA,FSYS
FSYS    0
        JMS*    .DA
        JMP     .+7
TABLE   0
SYNFL   0
SYNT2   0
N       0
SYWORC  0
SYBITC  0
        LAC     (SYNT1      /SYNT1 IS A TABLE WHERE SYNCHRONIZING
        DAC     PT2         /SEQUENCE IS DERIVED
        TAD     (1
        DAC     PT1
        DAC     PT3
        LAC     TABLE
        DAC     SYPS        /SYPS IS PS IN QUESTION
        LAC*    N
        RCL
        RCL
        TAD     TABLE
        DAC     TABEND      /TABLE'S END
        LAC     TABLE
        CMA
        TAD     (1
        DAC     TABBEG      /TABBEG = -(ADDRESS OF TABLE'S BEGINNING)
        DZM*    SYNFL
        DZM     SCT         /SCT = NUMBER OF STATES IN TABLE
        LAC     (1
        DAC     CT5         /CT5 IS STATE COUNT
        DZM*    PT1
        DZM*    PT2
        LAC     (MASK
        DAC     MASK
FS1     LAC*    SYPS
        SZA                 /IS ENTRY BLANK ?
        SKP                 /NO
        JMP     FS3         /YES
        ISZ     SCT         /INCREMENT SCT
        LAC     (-23

        TAD     CT5
        SPA                 /HAVE 18 STATES BEEN CHECKED ?
        JMP     FS2         /NO
        LAC     PT1         /YES; MOVE TO NEXT WORD
        TAD     (-1
        DAC     PT1
FS2     LAC     MASK
        TAD     CT5
        DAC     TMP6        /GET PROPER MASK
        LAC*    TMP6
        TAD*    PT1         /STORE IMPLICATED STATE
        DAC*    PT1
FS3     LAC     SYPS
        TAD     (4
        DAC     SYPS        /GO TO NEXT STATE
        ISZ     CT5         /UPDATE STATE COUNT
        LAC     TABEND
```

```
        CMA
        TAD     (1
        TAD     SYPS
        SPA                     /HAS END OF TABLE BEEN REACHED ?
        JMP     FS1             /NO
        LAC     SCT             /YES
        DAC     FS5
        TAD     (-1
        DAC     FS4
        CLL
        MUL
FS4     0
        LACQ
        CLL
        MUL
FS5     0
        LACQ
        RCR
        DAC     MLEV            /MLEV = (N-1)(N-1)N/2
        DZM     SYINP1          /SYINP1 IS AN INPUT
        DZM     SYINP2          /SYINP2 IS ALSO AN INPUT
        LAC     PT2
        TAD     (2
        DAC     PT2
        LAC     PT3
        TAD     (2
        DAC     PT3
        DZM     LEV             /LEV = # OF INPUTS IN SEQUENCE
FS6     LAC     (1
        DAC     CT5             /STATE COUNT = 1
        LAC     PT2
        TAD     (-1             /SYPS NOW REPRESENTS A NODE IN
        DAC     SYPS            /THE SYNCHRONIZING TREE
        LAC     SYINP1
        DAC*    PT2             /STORE THE NEXT INPUT
        DZM*    PT3             /CLEAR REST OF NODE STORAGE AREA
        DZM     CT5F            /CT5F INDICATES IF STATE NUMBER > 18
FS7     LAC     (-23
        TAD     CT5
        SPA                     /IS STATE NUMBER > 18 ?
        JMP     FS8             /NO
        LAC     CT5F            /YES
        SZA                     /IS STATE NUMBER = 18 ?
        JMP     FS8             /NO
        ISZ     CT5F            /YES
        LAC     SYPS
        TAD     (-1
        DAC     SYPS            /MOVE TO OTHER WORD OF NODE STORAGE AREA
FS8     LAC     MASK
        TAD     CT5             /MASKCT POINTS TO PROPER MASK
        DAC     MASKCT          /USED TO SELECT A STATE
        LAC*    SYPS            /GET ONE WORD OF NODE
        AND*    MASKCT          /IS STATE INDICATED BY CT5 A MEMBER OF
        SZA                     /THE NODE BEING WORKED WITH ?
        SKP                     /YES
        JMP     FS11            /NO
        LAC     CT5
```

```
        TAD     (-1
        RCL
        RCL
        TAD     SYINP2
        TAD     TABLE
        DAC     TMP6
        LAC*    TMP6        /GET IMPLICATED STATE
        TAD     TABBEG
        RCR
        RCR
        TAD     (1
        DAC     TMP7        /STORE IMPLICATED STATE
        LAC     (-23
        TAD     TMP7
        SPA                 /IS IMPLICATED STATE # > 18 ?
        SKP                 /NO
        JMP     FS9         /YES
        LAC     PT3
        DAC     PT1         /PT1 CONTAINS ADDRESS OF SECOND WORD OF NODE
        JMP     FS10
FS9     LAC     PT2
        DAC     PT1         /PT1 CONTAINS ADDRESS OF FIRST WORD OF NODE
FS10    LAC     MASK
        TAD     TMP7
        DAC     MASKCT      /CORRECT MASKCT
        LAC*    PT1         /GET NODE
        AND*    MASKCT
        SZA                 /HAS IMPLICATED STATE ALREADY BEEN SELECTED ?
        JMP     FS11        /YES
        LAC*    MASKCT      /NO
        TAD*    PT1
        DAC*    PT1         /STORE IMPLICATED STATE IN NODE
FS11    ISZ     CT5         /UPDATE STATE COUNT
        LAC     (-25
        TAD     CT5
        SPA                 /HAVE ALL IMPLICATED STATES BEEN FOUND ?
        JMP     FS7         /NO
        DZM     NSCT        /YES; NSCT = # OF IMPLICATED STATES AT A NODE
        LAC*    PT2         /GET FIRST WORD OF NODE
        AND     (3          /IT CONTAINS ONLY 2 BITS OF INFORMATION
        DAC     TMP8
        LAC     (-2
        DAC     CT7
        DAC     CT6         /NODE IS STORED IN 2 WORDS
FS12    LAC     TMP8        /GET NODE
        RCR                 /SHIFT INFORMATION INTO LINK BIT
        DAC     TMP8
        SZL                 /IS THE STATE IMPLICATED ?
        ISZ     NSCT        /YES
        ISZ     CT7         /NO; HAVE ALL STATES BEEN CHECKED ?
        JMP     FS12        /NO
        ISZ     CT6         /YES; HAVE BOTH WORDS OF NODE BEEN CHECKED ?
        JMP     FS17        /NO
        ISZ     LEV         /YES
        LAC     NSCT
        TAD     (-1
        SZA                 /DOES THE NODE CONTAIN ONLY ONE STATE ?
```

```
        SKP             /NO
        JMP     FS23    /YES
        LAC     (SYNT1  /TMP9 IS A POINTER USED TO STEP THROUGH THE NODES
        DAC     TMP9    /ALREADY OBTAINED FOR COMPARISON PURPOSES
FS13    LAC*    TMP9
        AND     (3
        DAC     TMP10
        LAC*    PT2
        AND     (3
        SAD     TMP10   /HAS PRESENT NODE ALREADY BEEN IMPLICATED ?
        JMP     FS15    /MAYBE
        ISZ     TMP9    /NO
FS14    ISZ     TMP9
        LAC     TMP9
        SAD     PT2     /HAVE ALL NODES BEEN COMPARED ?
        JMP     FS16    /YES
        JMP     FS13    /NO
FS15    ISZ     TMP9
        LAC*    TMP9
        SAD*    PT3     /HAS PRESENT NODE ALREADY BEEN IMPLICATED ?
        JMP     FS18    /YES
        JMP     FS14    /NO
FS16    LAC     NLEV
        CMA
        TAD     (1
        TAD     LEV
        SPA             /HAS SEQUENCE LENGTH REACHED MAXIMUM ?
        JMP     FS20    /NO
        JMP     FS18    /YES
FS17    LAC     (-22
        DAC     CT7     /RESET CT7 AND CONTINUE IMPLICATED STATE COUNT
        LAC*    PT3
        DAC     TMP8    /GET REST OF NODE(NODE IS STORED IN 2 WORDS)
        JMP     FS12
FS18    LAC*    PT2     /INPUT IS HIGH ORDER BIT OF WORD
        AND     (400000
        SZA             /IS INPUT "0" ?
        SKP             /NO
        JMP     FS21    /YES
FS19    LAC     PT2
        TAD     (-2
        DAC     PT2     /CORRECT POINTER FOR FIRST WORD OF NODE STORAGE
        LAC     LEV
        TAD     (-1
        DAC     LEV     /CORRECT NUMBER OF INPUTS COUNT
        LAC     PT3
        TAD     (-2
        DAC     PT3     /CORRECT POINTER FOR SECOND WORD OF NODE STORAGE
        LAC     (SYNT1
        CMA
        TAD     (1
        TAD     PT2
        SZA             /HAVE ALL SEQUENCES BEEN TRIED ?
        JMP     FS18    /NO
        JMP*    FSYS    /YES
FS20    LAC     SYINP1
        SZA             /IS INPUT "0" ?
```

```
        SKP             /NO
        JMP     FS22    /YES
        DZM     SYINP1
        DZM     SYINP2
        JMP     FS22
FS21    LAC     (2
        DAC     SYINP2
        LAC     (400000
        DAC     SYINP1
        LAC     LEV
        TAD     (-1
        DAC     LEV     /CORRECT NUMBER OF INPUTS COUNT
        JMP     FS6
FS22    ISZ     PT2     /MOVE TO NEXT NODE STORAGE AREA
        ISZ     PT2
        ISZ     PT3     /MOVE TO NEXT NODE STORAGE AREA
        ISZ     PT3
        JMP     FS6
FS23    LAC     (1
        DAC*    SYNFL   /A SEQUENCE HAS BEEN FOUND
        LAC     LEV
        DAC*    SYBITC  /STORE LENGTH OF SEQUENCE
        TAD     (-1
        DAC     MLEV    /UPDATE MLEV(MAXIMUM LENGTH OF SEQUENCE)
        LAC     LEV
        CMA
        TAD     (1
        DAC     CT4     /CT4 CONTAINS NUMBER OF INPUTS IN THE SEQUENCE
        LAC     PT2
        DAC     CT7     /CT7 CONTAINS ADDRESS OF LAST INPUT
        LAC     SYNT2   /SYNT2 IS ADDRESS WHERE SEQUENCE IS STORED
        DAC     SYIWOR  /SYIWOR CONTAINS ADDRESS WHERE INPUTS ARE STORED
        LAC     (1
        DAC*    SYWORC  /SYWORC CONTAINS NUMBER OF WORDS OF INPUTS
FS24    LAC     (-22
        DAC     CT6
        DZM*    SYIWOR
FS25    LAC*    SYIWOR
        RCR
        DAC*    SYIWOR
        LAC*    CT7
        AND     (400000 /GET NEXT INPUT
        TAD*    SYIWOR
        DAC*    SYIWOR  /STORE INPUT
        ISZ     CT4     /HAVE ALL INPUTS BEEN STORED ?
        JMP     FS26    /NO
        JMP     FS19    /YES
FS26    LAC     CT7
        TAD     (-2
        DAC     CT7     /MOVE TO NEXT INPUT
        ISZ     CT6     /HAVE 18 INPUTS BEEN STORED IN ONE WORD ?
        JMP     FS25    /NO
        LAC     (-22    /YES
        TAD*    SYBITC  /UPDATE SYBITC
        DAC*    SYBITC
        ISZ     SYIWOR  /MOVE TO NEXT WORD FOR STORAGE OF INPUTS
        ISZ*    SYWORC  /UPDATE WORD COUNT
```

```
                JMP         FS24
PT1
PT2
PT3
CT4
CT5
CT6
CT7
SYPS
TABEND
TABBEG
MLEV
LEV
SYIWOR
MULTC
MULTR
MULTR2
SYINP1
SYINP2
TMP6
TMP7
TMP8
TMP9
TMP10
MASKCT
NSCT
SCT
CTSF
MASK
                000001
                000002
                000004
                000010
                000020
                000040
                000100
                000200
                000400
                001000
                002000
                004000
                010000
                020000
                040000
                100000
                200000
                400000
                000001
                000002
                000004
SYNT1           .BLOCK      7300
                .END        FSYS
```

APPENDIX II

Appendix II contains solutions to two examples as solved by the program listed in Appendix I.

Example 1 uses the following machine as the unknown machine. The machine was assumed to be initially in state A.

PS	NS		z
	x = 0	x = 1	
A	B	A	1
B	A	C	0
C	D	C	0
D	C	B	1

00
01
00
01
00
01
11
11
11
11
00
10

PS	NS		Z
	0	1	
S 1	S 2	S 1(3)	0
S 2	S 1(1)	S 2(2)	1

PS	NS		Z
	0	1	
S 1	S 2	S 1(3)	0
S 2	S 1(1)	S 2(2)	1
S 3			0
S 4			1

01
00
10
10

PS	NS		Z
	0	1	
S 1	S 2	S 3(3)	0
S 2	S 1(1)	S 2(2)	1
S 3	S 2(4)	S 1(5)	0
S 4			1

Example 1:

Example 1. (Continued)

PS	NS		Z
	0	1	
S 1	S 2	S 3(3)	0
S 2	S 1(1)	S 2(2)	1
S 3	S 2(4)	S 3(5)	0
S 4			1

10
01
10

PS	NS		Z
	0	1	
S 1	S 2	S 3(3)	0
S 2	S 1(1)	S 2(2)	1
S 3	S 4(4)	S 1(6)	0
S 4	S 1(5)	S 1(7)	1

01
00
10
01
10
01
00
01
00
01
11
00
10
01
00
10
01
00
10
01
10
01
00

Example 1. (Continued)

```
01
00
01
00
10
01
10
01
00
01
11
00
01
11
00
10
01
00
10
01
00
01
10
```

PS	NS		Z
	0	1	
S 1	S 2	S 3(3)	0
S 2	S 1(1)	S 2(2)	1
S 3	S 4(4)	S 3(6)	0
S 4	S 3(5)	S 1(7)	1

```
10
10
01
10
10
01
10
01
11
11
00
10
```

Example 1. (Continued)

```
10
01
10
10
10
10
01
10
10
01
10
01
00
10
10
01
10
10
01
10
01
11
11
11
00
10
10
01
10
10
01
10
10
10
10
10
01
10
10
01
00
10
10
01
10
10
01
10
```

Example 1. (Continued)

PS	NS		X
	0	1	
S 1	S 2	S 3(3)	0
S 2	S 1(1)	S 2(2)	1
S 3	S 4(4)	S 3(6)	0
S 4	S 3(5)	S 1(7)	1

OF INPUTS 000175
OF INCORRECT ENTRIES 000017
OF INPUTS/# OF INCORRECT ENTRIES = 000010 + 000005/000017

END

Example II uses the following machine as the unknown machine. The machine was assumed to be initially in state A.

PS	NS		z
	x = 0	x = 1	
A	F	A	0
B	F	A	1
C	H	C	0
D	H	C	1
E	J	G	1
F	B	C	0
G	B	C	1
H	D	E	0
I	D	E	1
J	E	I	0

```
00
01
00
01
00
01
10
10
10
10
```

PS	NS		Z
	0	1	
S 1	S 2	S 1(3)	0
S 2	S 1(1)	S 1(2)	1

```
10
00
```

PS	NS		Z
	0	1	
S 1	S 2	S 3(3)	0
S 2	S 1(1)	S 1(2)	1
S 3	S 1(5)	S 3(4)	0
S 4			1

```
10
10
00
01
10
10
00
11
```

Example 2.

Example 2. (Continued)

PS	NS		Z
	0	1	
S 1	S 2	S 2(7)	0
S 2	S 1(1)	S 3(2)	1
S 3	S 5(5)	S 5(3)	0
S 4			1
S 5	S 1(6)	S 3(4)	0
S 6			1

00
11
11

PS	NS		Z
	0	1	
S 1	S 2	S 4(7)	0
S 2	S 1(1)	S 3(2)	1
S 3	S 5(5)	S 5(3)	0
S 4	S 1(10)	S 2(11)	1
S 5	S 1(6)	S 3(4)	0
S 6			1

00
11
11
00
01
00
11
11
00
11
01
10
00
01
00

Example 2. (Continued)

PS	NS		Z
	0	1	
S 1	S 2	S 6(12)	0
S 2	S 1(1)	S 3(2)	1
S 3	S 5(4)	S 3(3)	0
S 4	S 1(15)	S 5(10)	1
S 5	S 7(6)	S 5(5)	0
S 6	S 4(14)	S 2(13)	1
S 7	S 4(7)	S 2(11)	0
S10			1

01
00
01
00
11
00
11

PS	NS		Z
	0	1	
S 1	S 2	S10(13)	0
S 2	S 1(1)	S 3(2)	1
S 3	S 5(4)	S 3(3)	0
S 4	S 7(16)	S 5(10)	1
S 5	S 7(6)	S 5(5)	0
S 6	S 1(12)	S 4(17)	1
S 7	S 4(7)	S 6(11)	0
S10	S 4(15)	S 6(14)	1

10
00
11
11
10
00
11
00
01
11

Example 2. (Continued)

PS	NS		Z
	0	1	
S 1	S 2	S 3(3)	0
S 2	S 1(1)	S 1(2)	1
S 3	S 5(5)	S 3(4)	0
S 4	S 7(20)	S 5(11)	1
S 5	S 7(7)	S 5(6)	0
S 6	S11(13)	S 4(21)	1
S 7	S 4(10)	S 6(12)	0
S10	S 4(17)	S 6(15)	1
S11	S 6(16)	S10(14)	0
S12			1

10
10
00
01
10
10
00
11
11
01

PS	NS		Z
	0	1	
S 1	S 2	S 3(3)	0
S 2	S 1(1)	S 1(2)	1
S 3	S 5(5)	S 3(4)	0
S 4	S 7(20)	S 5(11)	1
S 5	S 7(7)	S 5(6)	0
S 6	S11(13)	S12(21)	1
S 7	S 4(10)	S 6(12)	0
S10	S 4(17)	S 6(15)	1
S11	S 6(16)	S10(14)	0
S12	S 2(23)	S 5(22)	1

10
10
10
10
00

Example 2. (Continued)

```
10
10
10
00
11
11
01
00
01
10
10
10
10
00
10
10
10
00
11
11
01
00
10
10
10
10
10
00
11
```

PS	NS		Z
	0	1	
S 1	S 2	S 5(5)	0
S 2	S 1(1)	S 3(2)	1
S 3	S 1(4)	S 3(3)	0
S 4	S 7(21)	S 1(11)	1
S 5	S 7(7)	S 5(6)	0
S 6	S11(13)	S12(22)	1
S 7	S 4(10)	S 6(12)	0
S10	S12(17)	S 6(15)	1
S11	S 6(16)	S10(14)	0
S12	S 2(23)	S 5(20)	1

Example 2. (Continued)

```
11
01
10
00
10
10
10
10
00
01
10
00
```

PS	NS		Z
	0	1	
S 1	S 2	S 5(5)	0
S 2	S 1(1)	S 3(2)	1
S 3	S 1(4)	S 3(3)	0
S 4	S 7(20)	S 5(11)	1
S 5	S 7(7)	S 5(6)	0
S 6	S11(13)	S12(21)	1
S 7	S 4(10)	S 6(12)	0
S10	S 4(17)	S 6(15)	1
S11	S 6(16)	S10(14)	0
S12	S 2(23)	S 5(22)	1

```
11
00
11
01
10
10
10
10
00
11
11
01
00
01
10
00
10
```

Example 2. (Continued)

```
00
11
11
10
10
00
11
11
01
00
10
10
00
11
00
11
11
11
10
00
11
11
01
00
10
00
11
00
11
11
11
10
00
11
11
01
10
10
00
10
00
11
11
10
10
00
11
```

Example 2. (Continued)

```
11
01
10
00
10
00
11
00
11
11
11
10
00
11
11
01
10
10
10
00
10
00
11
11
10
10
00
01
00
11
00
11
01
10
10
10
10
00
01
10
10
00
11
00
11
11
11
10
```

Example 2. (Continued)

```
00
11
00
11
01
10
10
10
10
10
10
00
11
00
11
11
11
10
00
11
00
11
01
10
00
11
11
10
10
00
11
11
10
00
11
00
11
11
11
10
00
01
10
00
11
00
11
```

Example 2. (Continued)

```
11
11
10
00
11
11
01
10
00
10
10
10
10
00
11
00
11
01
10
00
11
11
11
11
10
00
11
00
11
11
11
01
10
00
10
10
10
10
00
11
00
01
11
01
10
00
10
10
10
```

Example 2. (Continued)

```
10
00
11
00
11
11
00
11
01
10
10
10
10
00
11
11
01
10
00
10
00
11
11
10
10
00
11
11
10
```

PS	NS		Z
	0	1	
S 1	S 2	S 5(5)	0
S 2	S 1(1)	S 3(2)	1
S 3	S 1(4)	S 3(3)	0
S 4	S 7(20)	S 5(11)	1
S 5	S 7(7)	S 5(6)	0
S 6	S11(13)	S12(21)	1
S 7	S 4(10)	S 6(12)	0
S10	S 4(17)	S 6(15)	1
S11	S 6(16)	S10(14)	0
S12	S 2(23)	S 5(22)	1

Example 2. (Continued)

```
# OF INPUTS      000546
# OF INCORRECT ENTRIES      112651
# OF INPUTS/# OF INCORRECT ENTRIES = 000000 + 000546/112651


END
```

LIST OF REFERENCES

1. Booth, T. L., Sequential Machines and Automata Theory, New York, New York: John Wiley and Sons, Inc., 1967.

2. Kohavi, Z., Switching and Finite Automata Theory, New York, New York: McGraw-Hill, Inc., 1970.