THE UNIVERSITY OF MICHIGAN

SYSTEMS ENGINEERING LABORATORY

Department of Electrical Engineering
College of Engineering

SEL Technical Report No. 20

COMPUTER PROGRAMS DEALING WITH FINITE STATE MACHINES: PART II

by

Thomas F. Piatkowski

July 1967

This research was supported by United States Air Force Contract AF 30(602)-3546.

TABLE OF CONTENTS

Section		Page
I.	Introduction	1
II.	Conventions	3
III.	Simple Adaptive Diagnosing	4
IV.	Finding an Equivalent Regular Expression	32
V.	Displaying the Lattice of SP Partitions	49
VI.	Bibliography	71

I. Introduction

During the past several months I have been able to program a number of algorithms dealing with finite-state machines; the purposes of this effort being: (1) pedagogical application, (2) the stimulation of insight, and (3) the generation of new questions and approaches. Out of a set of thirty-eight more or less different problems dealing with finite state machines, I selected seven representatives; namely:

1) simulate a given machine,
2) determine if a given machine is strongly connected,
3) determine the state equivalence classes and minimal form for a given machine,
4) determine the automorphisms and their group for a given machine,
5) determine a shortest simple adaptive diagnosing experiment for a given machine and admissible set,
6) determine an equivalent regular expression for a given machine,
7) exhibit the lattice of SP partitions for a given machine.

This report, the second of two, describes the programs treating problems five through seven; the first report treating problems one through four appeared previously.

Concurrent with my own programming efforts, I attempted to document existing programs in this area; the results of my search are included in the bibliography.

The first two programs contained in this report are written in 7090 SNOBOL for execution in the batch mode; the third program is written in 360 FORTRAN IV for execution in the time-sharing mode on MTS, the Michigan Terminal System. A conscious effort has been made to make them straightforward and easy to use.

II. Conventions

We will deal with N, P, Q machines where

N = the number of states in state set S,

P = the number of symbols in input alphabet A_I,

Q = the number of symbols in output alphabet A_O.

We will only consider machines coded such that

$$S = \{1, 2, \ldots, N\}$$

$$A_I = \{1, 2, \ldots, P\}$$

$$A_O = \{1, 2, \ldots, Q\}$$

A machine is a system $M = \langle S, A_I, A_O, FS, FZ \rangle$ where S, A_I, and A_O are as above and where FS, the next state function, maps $S \times A_I \rightarrow S$ and FZ, the output function, maps $S \times A_I \rightarrow A_O$ for Mealy machines and $S \rightarrow A_O$ for Moore machines.

It is assumed that the reader is familiar with the necessary theoretical concepts of finite-state machine theory [See 1, 2, and 3] and with the programming languages SNOBOL and FORTRAN IV.

The first two programs were written to run in SNOBOL under the University of Michigan Executive System for the IBM 7090 Computer as it existed in May 1967. The third program was written to run in FORTRAN IV under the University of Michigan Terminal System for the IBM 360 as it existed in July 1967.

III. Simple Adaptive Diagnosing

A. Program Name: SAD

B. Purpose: To analyze a given finite-state machine and admissible set and determine if a simple adaptive diagnosing experiment exists; if one does, display a representative from the class of shortest such experiments (the length of an experiment is defined here as the length of the longest input sequence required - i. e. the worst case).

C. Method: The method of solution consists in properly creating and interpreting a diagnosing tree. The tree has AM, the admissible set, associated with its root node. Branches and nodes are added and deleted by alternate application of growing and pruning algorithms until each branch is appropriately terminated; at this point the tree structure will indicate if an adaptive diagnosing experiment exists and if so display the procedure for carrying it out.

Growing Algorithm: Let M be the machine under consideration.

Let S_0, the source set, $= \{s_1, s_2, \ldots, s_r\} \subseteq S$ represents a range of uncertainty in our knowledge of the present state of M; i. e., we know that the present state of M is one of the states in S_0 but no more.

Associate a node with S_0; call this node the source node. From the source node extend P branches, each corresponding to a symbol in A_I. Terminate each branch with a distinct node and label each node with the corresponding input; call these new nodes the input nodes. From each input node extend Q branches, each corresponding to a symbol in A_0. Terminate each branch with a distinct node and label the node with the corresponding output; call these new nodes the successor nodes.

Each successor node is derived from the source node by a unique input, I, and output, J; therefore we can speak unambiguously about the I/J successor node of the source node for all $1 \leq I \leq P$ and $1 \leq J \leq Q$. With each I/J successor node associate an I/J successor set as follows:

1) I/J successor set of $S_0 = \{FS(s, I) \mid s \in S_0,\ FZ(s, I) = J\}$. In other words, the I/J successor set of S_0 is the set of all states in M such that each can be reached from a state in S_0 on the single input I with accompanying output J. Contrary to the usual convention in set theory, however, explicitly indicate multiple non-distinct entries

in any successor set by listing these elements more than once; i. e., if two or more states in S_0 go into the same state s' on input I with output J, then s' will appear two or more times in the I/J successor set of S_0. For example, Figure 2 illustrates the above construction for machine M1 (Figure 1) and source set $\{1, 2, 3\}$.

The growing algorithm displays the possibilities for altering the range of uncertainty in our knowledge of the present state of the machine by applying a single input and monitoring the attending output; i. e., if the present state of the machine is known to be in the source set then if input I causes the attending output J, the new state of the machine must be in the I/J successor set of the source set.

Successor Node Evaluation: Application of the growing algorithm results in $P \times Q$ successor nodes for each source node. The successor set corresponding to each successor node can be either

1) empty
2) a singleton
3) multiple with distinct entries, or
4) multiple with at least one repeated entry.

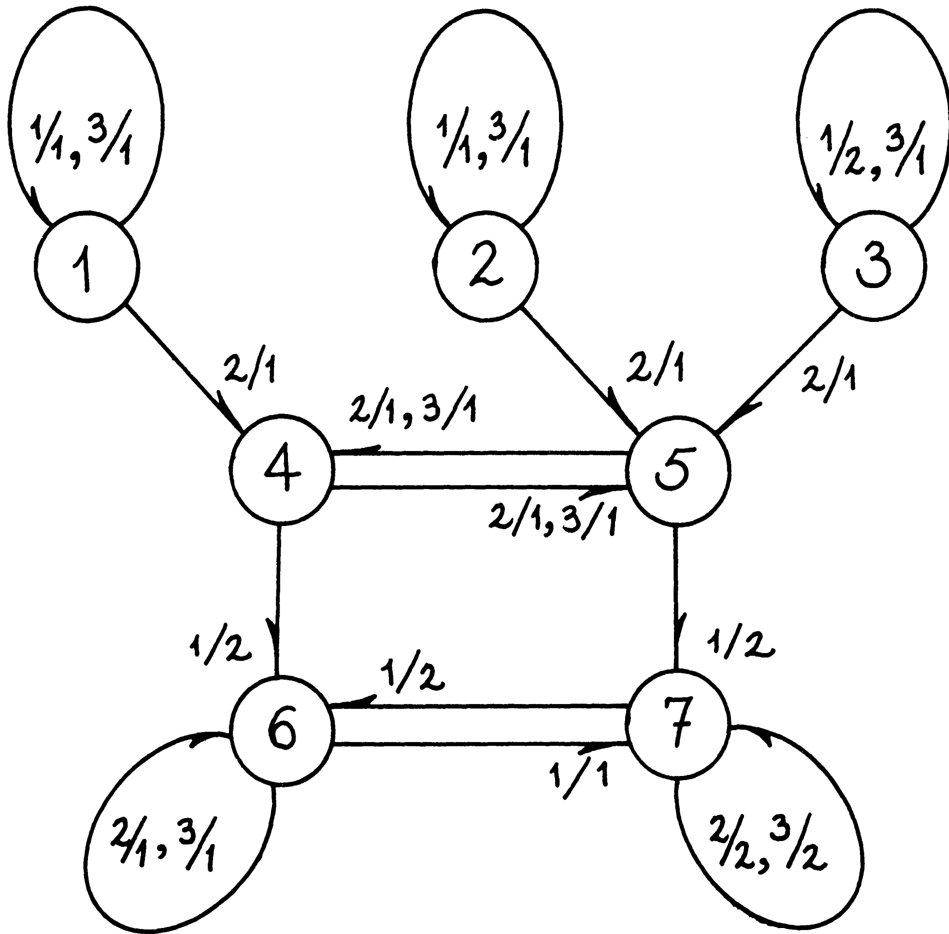


Figure 1. M1, a 7,3,2 Machine

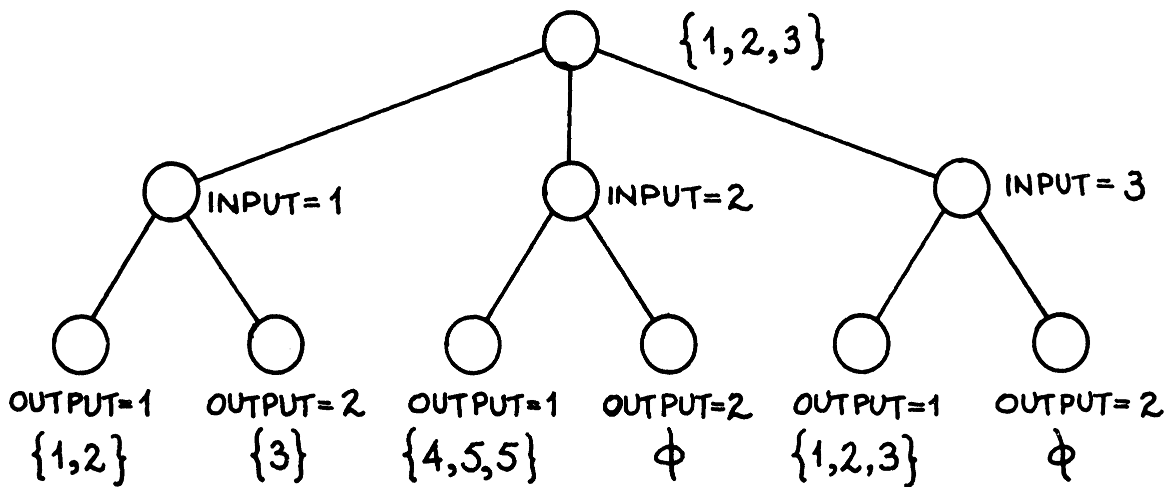


Figure 2. Derivation of All I/J Successors of Source Set $\{1, 2, 3\}$ for M1

An empty I/J successor set implies the impossibility of getting an output J when input I is applied to any state in the source set. Such successor nodes and their corresponding branches are deleted from the tree without consequence.

A singleton I/J successor set implies that a unique state in the source set yields output J in response to input I. In the event this unique state was indeed the initial state of M the inputting of I with attendent outputting of J would identify the fact. Nodes corresponding to singleton sets are terminal in that no further experimenting is needed to identify the machine's present state and its predecessor state in the source set.

A multiple distinct I/J successor set implies that if the initial state of M is in the source set and the inputting of I is attended by output J, then the range of uncertainty in the new present state of M is shifted to the I/J successor set with no introduction of ambiguity in the predecessor state; i. e., each state in the I/J successor set has a unique predecessor in the source set and therefore the ability to diagnose the successor set would ensure the ability to diagnose the initial state in the source set, conditional on the original I, J input/output pair. Nodes corresponding to multiple distinct I/J successor sets are not terminal since they indicate some uncertainty still exists in our knowledge of the machine's present state;

in general this range of uncertainty is different and possibly reduced from the range of uncertainty in the source set.

A multiple repeated I/J successor set implies that at least two states of the source set yield identical outputs and move to identical successor states in response to input I; such a condition precludes the distinguishing of the initially distinct states by any further experimenting; thus input I must not be applied to M when the range of uncertainty is the source set since doing so will, in general, prohibit a successful diagnosis. For this reason nodes corresponding to multiple repeated I/J successor sets are terminal.

Node Logic: The source set is diagnosable iff there exists at least one input I such that for all outputs J, the I/J successor sets are either empty or diagnosable. In other words, source nodes act as OR gates and input nodes as AND gates. This may be indicated explicitly in a graphical presentation of the diagnosis tree by drawing source nodes as ⊕ and input nodes as ⊙.

Furthermore, singleton sets, being always diagnosable, are interpreted as terminal TRUE nodes; multiple repeated successor sets, being never diagnosable, are interpreted as terminal FALSE nodes and both TRUE and FALSE terminal nodes have the usual implication on the truth value of preceeding AND and OR gates.

Erecting and Pruning the Tree: The tree is created by erecting successive levels of successor nodes corresponding to all input strings of length 1, then 2, then 3, etc. .... The process is initiated by associating AM, the admissible set, with the single source node at level zero. The first level of successor nodes is erected from the root and then, in turn, provides the source nodes for erection of the second level of successor nodes and so on. In general each level of successor nodes will contain a number of terminal nodes - some TRUE, some FALSE; as each terminal node is created the implication of its truth value is projected as far as possible back through the tree toward the root node. An efficiency of computation is effected by accompanying the backward implication of each new terminal node with the following pruning procedures:

1) If an AND node is TRUE then its preceeding OR node is also TRUE and all portions of the tree descending from the OR node save for the branch to the TRUE AND may be deleted without affecting the truth value of the OR node.

2) If an OR node is FALSE its implication of the truth value FALSE backward toward the root node will halt either at the root node itself or at some AND node which we will call the parent node (a FALSE OR always implies the preceeding AND is FALSE thus backward

implication cannot halt at an OR node unless it be the root). In the latter event delete the parent node, its descendent subtree and the branch descending into the parent node; none of these deletions affect the evaluation of the OR node preceeding the parent node.

In general a single TRUE terminal node may induce several applications of rule 1 whereas a single FALSE terminal node will induce only a single application of rule 2.

Cycles: There is one additional condition under which a successor node is terminal FALSE; i.e. when its associated set is identical to that of one of its predecessors (not necessarily the immediate predecessor); this follows from the observation that no diagnostic purpose is served by allowing an input/output sequence that causes the range of uncertainty to repeat. Since each input/output sequence corresponds uniquely to its final successor node, the negation of a particular cyclic successor node will eliminate from consideration precisely the proper input/output sequence without of itself eliminating any other sequences. A cyclic terminal node has the same backward implications in the tree as any other FALSE node.

<u>Addition of Initial State Information to the State Sets:</u> Every state set corresponding to a node in the tree explicitly denotes the range of uncertainty in the present state of the machine after a particular input/output sequence. No explicit information is given regarding the range of uncertainty in the initial state of the machine. This information may be explicitly added in the following manner: each state s in the state set under consideration is an I/J successor of some unique initial state s' in the admissible AM; replace the element s in the state set by the state pair s. s' — thus the sets associated with nodes in the tree will be sets of state pairs, the first state in each pair denoting a possible present state and the second denoting the corresponding initial state. For example, in Figure 2 the root node would now have the associated state set $\{1.1,\ 2.2,\ 3.3\}$ and the 2/1 successor of the root node the associated state set $\{4.1,\ 5.2,\ 5.3\}$, etc. .... .

The explicit denotation of the initial state associated with each present state is conveniently expedited by modifying equation (1) to read

(2) $\quad$ I/J successor set of $S_0 = \{FS(s, I).\ s' \mid s.\ s' \in S_0,\ FZ(s, I) = J\}$

Thus the initial state information is created as each successor set is created with reference being made only to the source node involved.

The explicit display of initial state information in the state sets in no way influences the growth of the tree; i. e. the evaluation of terminal nodes, whether singletons, multiple repeated or cyclic depends only on present state information.

<u>Termination and Interpretation of the Computation:</u> Construction of the tree is carried out until the truth value of the root node is established; if this value is FALSE no adaptive diagnosing experiment for the given machine and admissible set is possible; if this value is true then an adaptive diagnosing experiment does exist and is carried out as follows:

1) Obtain a copy of the given machine with initial state in the admissible set.
2) Let the first source node be the root node of the tree; i. e. before any input has been applied to the machine the present state and initial state are identical. By assumption this node is TRUE.
3) The source set displays the current range of uncertainty in the present state and the associated initial state of the machine. If the source set is a singleton the present and initial states of the machine are known precisely, the experiment is complete, halt; if the source set is not a singleton go to 4).

4) The pruning algorithm and node logic guarantee that each non-terminal OR node in the tree has a unique TRUE successor input node; in particular the current source node has a unique TRUE successor input node with associated input I. Apply input I to the machine and note the output J. Let the I/J successor of the source node become the new source node; this node must be TRUE or else node logic would require the preceeding AND node to be FALSE which would be a contradiction. Go to 3).

For example, Figure 3 displays the completed diagnosing tree for machine M1 with AM = S; the positive result of this example ensures that <u>any</u> admissible set is diagnosable in M1.

D. Language and System: The program is written in SNOBOL (31 December 1965 version) for execution on the University of Michigan Executive System for the IBM 7090 computer as it existed in May 1967.

E. Input: The input deck consists of an arbitrary number of sub-decks; each sub-deck names and describes a finite-state machine and lists one or more admissible sets to be diagnosed for the given machine. The

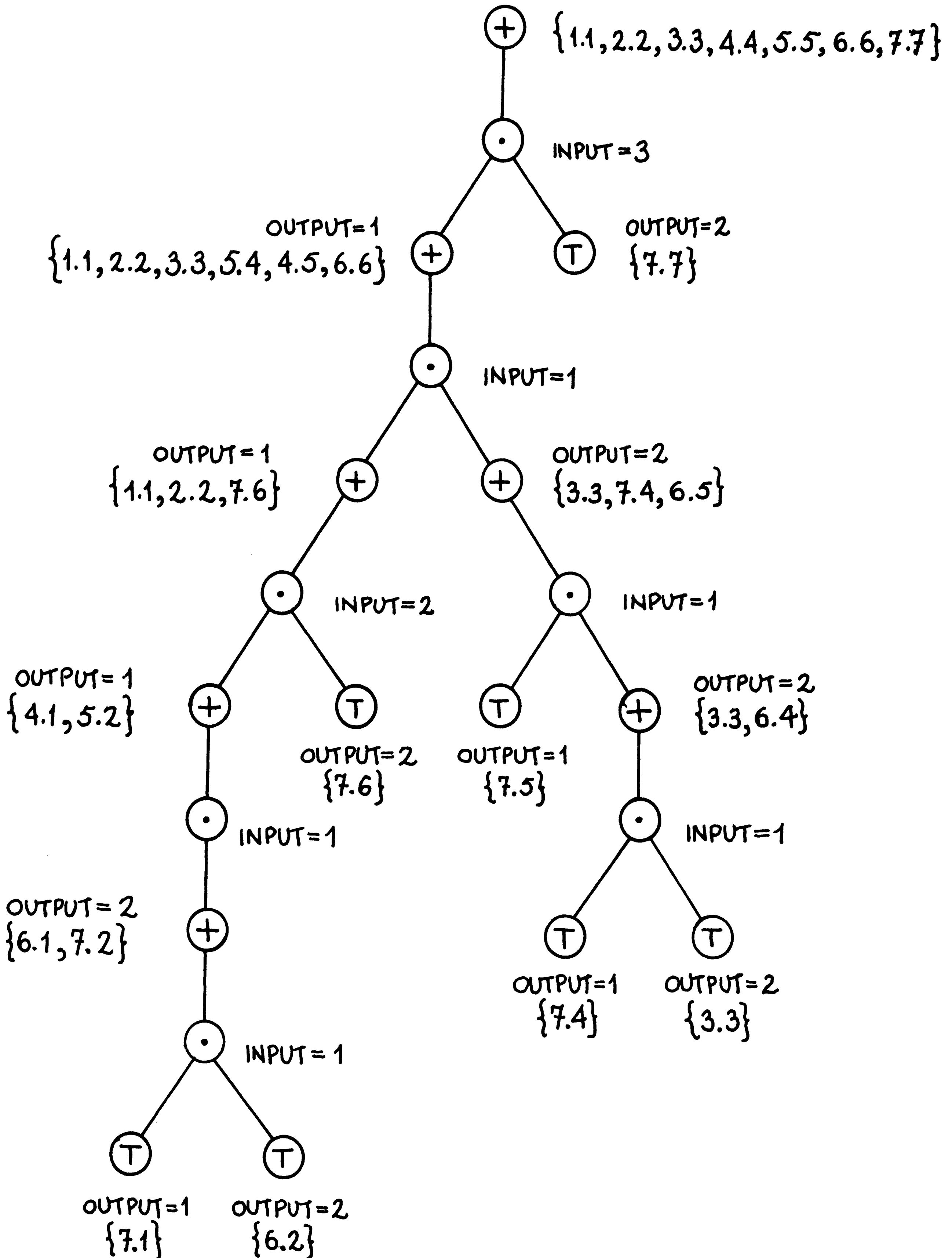


Figure 3. Completed Diagnosing Tree for M1,
$AM = \{1, 2, 3, 4, 5, 6, 7\}$

make-up of each sub-deck and the card formats are as follows:

1) Name card - up to 80 columns are available for any string of alphameric characters identifying the machine by name; trailing blanks are ignored. This card must head each sub-deck even if the name is blank.

2) N, P, Q card - gives the values of N, P, Q for the machine; three integers separated by commas, no blanks, left justified on the card.

3) Transition table cards - N cards, each of the following form:

$s, s_1/j_1, s_2/j_2, \ldots, s_P/j_P$

where s, s_i, j_i are integers separated by comma and slash as above and all data is left justified with no blanks and where

$s \in S$

$s_i = FS(s, i)$

$j_i = FZ(s, i)$

There should be precisely one transition table card for each state in M; the transition table cards may be in any order provided the last card has $s = N$. The transition cards provide the capability of describing any Mealy machine;

Moore machines may be described by the artifice of making FZ(s, i) independent of i.

4) AM Cards - an arbitrary number of cards, each specifying an admissible set to be tried on M and each in the following form: $(s_1, s_2, \ldots, s_m)$, where the s_i are integers separated by commas, s_1 is preceeded by an open parenthesis in card column 1 and s_m $(m \leq N)$ is succeeded by a closed parenthesis; the parentheses enclose no blanks. Each sub-deck must contain at least one admissible set and each admissible set at least one element.

Restrictions: The only limitations on the size of machines (i. e., N, P, Q) or admissible sets that this program will handle are those imposed by the SNOBOL compiler itself on string length, number of strings, etc. .... However, the input routines for this program expect to find each of the following data entries on a single card

1) the machine name
2) the N, P, Q data
3) each row of the transition table
4) each admissible set

If multiple card input is required the input routines can be

rewritten; no change should be necessary in the body of the program.

F. Operation and Output: The program works through the data deck sub-deck by sub-deck and for each will output a description of the corresponding machine followed by a result for each admissible set. If no adaptive experiment exists for a particular admissible set a comment to that effect is made; if an adaptive experiment does exist then the diagnosing tree is listed in the following manner: Each of the non-terminal OR nodes are listed (each is identified by number). The zero node is the root node of the tree. Let X be the number of any OR node in the tree; listed with X in the output is 1) the associated state set indicating the range of uncertainty in the present and initial states, 2) the input I associated with the succeeding TRUE AND node (for each X this is unique), and 3) a list of successor OR nodes corresponding to the various outputs - this list has entries of the form

J/X' where J is an output symbol and X' is the number of the I/J successor to X. Terminal nodes will not be referenced by number, instead the entry 'STOP' followed by the initial and final states associated with that terminal node are given after the output.

The program does not reuse node numbers that have been pruned away - thus the completed tree and therefore the output list will, in general, not number OR nodes consecutively.

The output listing conveniently indicates how to perform a diagnosis; the procedure is to start at node zero, apply the associated input, observe output, move to indicated new node, apply associated input, etc. ... until a 'STOP' is encountered - at which point the diagnosis is made. At any point along the way the state set associated with each node gives the present range of uncertainty in both the present and initial states.

G. Sample Data

```
A1
7,3,2
1,1/1,4/1,1/1
2,2/1,5/1,2/1
3,3/2,5/1,3/1
4,6/2,5/1,5/1
5,7/2,4/1,4/1
6,7/1,6/1,6/1
7,6/2,7/2,7/2
(1,2,3)
(1,2,3,4,5,6,7)
A21,GILL,PAGE 106
10,2,2
1,5/1,1/1
2,6/1,2/1
3,7/2,3/1
4,8/2,4/1
5,9/1,6/1
6,9/1,7/2
7,6/2,10/1
8,7/1,10/1
9,9/1,9/2
10,10/2,10/1
(1,2,3,4)
(5,6,7,8)
(7)
```

H. Sample Output

```
SIMPLE ADAPTIVE DIAGNOSING

MACHINE NAME = M1

N = 7, P = 3, Q = 2

TRANSITION TABLES***STATE FOLLOWED BY LIST OF (NEXT STATE/OUTPUT) FOR SUCCESSIVE INPUTS

 1    1/1,4/1,1/1
 2    2/1,5/1,2/1
 3    3/2,5/1,3/1
 4    6/2,5/1,5/1
 5    7/2,4/1,4/1
 6    7/1,6/1,6/1
 7    6/2,7/2,7/2


MACHINE NAME = M1, ADMISSIBLE SET = (1,2,3)

THE FOLLOWING IS A SIMPLE ADAPTIVE DIAGNOSING EXPERIMENT FOR THE GIVEN MACHINE AND ADMISSIBLE SET

NO    SET      INPUT, FOLLOWED BY THE (OUTPUT/NEXT NO) LIST

 0    1.1,2.2,3.3    1
        1/1
        2/STOP(INIT = 3,FIN = 3)

 1    1.1,2.2     2
        1/3

 3    4.1,5.2     1
        2/4

 4    6.1,7.2     1
        1/STOP(INIT = 1,FIN = 7)
        2/STOP(INIT = 2,FIN = 6)


MACHINE NAME = M1, ADMISSIBLE SET = (1,2,3,4,5,6,7)

THE FOLLOWING IS A SIMPLE ADAPTIVE DIAGNOSING EXPERIMENT FOR THE GIVEN MACHINE AND ADMISSIBLE SET

NO    SET      INPUT, FOLLOWED BY THE (OUTPUT/NEXT NO) LIST

 0    1.1,2.2,3.3,4.4,5.5,6.6,7.7    3
        1/2
        2/STOP(INIT = 7,FIN = 7)

 2    1.1,2.2,3.3,5.4,4.5,6.6    1
        1/4
        2/5

 4    1.1,2.2,7.6    2
        1/8
        2/STOP(INIT = 6,FIN = 7)

 5    3.3,7.4,6.5    1
        1/STOP(INIT = 5,FIN = 7)
        2/13

 8    4.1,5.2     1
        2/19

13     3.3,6.4     1
        1/STOP(INIT = 4,FIN = 7)
        2/STOP(INIT = 3,FIN = 3)

19     6.1,7.2     1
        1/STOP(INIT = 1,FIN = 7)
        2/STOP(INIT = 2,FIN = 6)
```

```
SIMPLE ADAPTIVE DIAGNOSING

MACHINE NAME = A21,GILL,PAGE 106

N = 10, P = 2, Q = 2

TRANSITION TABLES***STATE FOLLOWED BY LIST OF (NEXT STATE/OUTPUT) FOR SUCCESSIVE INPUTS

 1    5/1,1/1
 2    6/1,2/1
 3    7/2,3/1
 4    8/2,4/1
 5    9/1,6/1
 6    9/1,7/2
 7    6/2,10/1
 8    7/1,10/1
 9    9/1,9/2
10     10/2,10/1


MACHINE NAME = A21,GILL,PAGE 106, ADMISSIBLE SET = (1,2,3,4)

THE FOLLOWING IS A SIMPLE ADAPTIVE DIAGNOSING EXPERIMENT FOR THE GIVEN MACHINE AND ADMISSIBLE SET

NO    SET      INPUT, FOLLOWED BY THE (OUTPUT/NEXT NO) LIST

 0    1.1,2.2,3.3,4.4    1
          1/1
          2/2

 1    5.1,6.2    2
          1/STOP(INIT = 1,FIN = 6)
          2/STOP(INIT = 2,FIN = 7)

 2    7.3,8.4    1
          1/STOP(INIT = 4,FIN = 7)
          2/STOP(INIT = 3,FIN = 6)


MACHINE NAME = A21,GILL,PAGE 106, ADMISSIBLE SET = (5,6,7,8)

NO SIMPLE ADAPTIVE DIAGNOSING EXPERIMENT EXISTS FOR THE GIVEN MACHINE AND ADMISSIBLE SET


MACHINE NAME = A21,GILL,PAGE 106, ADMISSIBLE SET = (7)

ADMISSIBLE SET IS A SINGLETON***NO EXPERIMENT NECESSARY


**** ALL INPUT DATA HAVE BEEN PROCESSED

        ELAPSED TIME --

        TOTAL          PROCESSING      EXECUTION      LOADING

        0' 21.8"       0'  2.0"        0' 14.3"       0'  5.5"
```

I. Important Program Variables: This portion of the report will use the conventions of SNOBOL to describe the contents of several important string variables appearing in the program.

OR nodes: each OR node is identified by a number, say X. Associated with each OR node is a string 'OR' X which contains information on that node and its function in the tree. The contents of 'OR' X will vary during execution as follows:

1) if node X is pruned from the tree then $('OR' X) is null,

2) if node X is terminal TRUE then

   $('OR' X) = 'F' FIN 'I' INIT 'θ' OUT

   where FIN = the associated final state

   INIT = the associated initial state

   OUT = the associated output symbol,

3) if node X is nonterminal TRUE then

   $('OR' X) = 'PT' SET 'θ' OUT 'S' Z','

   where SET = the state pair set corresponding to node X and will be of the form #.#,#.#,...,#.#,

   OUT = the associated output and

   Z = the number of the unique succeeding AND node

4) if node X is of undetermined truth value

$('OR' X) = 'L' LG 'PT' SET 'PR' Z 'S' LIST

where LG = the number of elements in SET (which is defined as in 3 above)

Z = the number of the immediate AND predecessor of X

LIST = a list (of the form #,#,...,#,) of the AND successors of X (all must be unvalued)

AND nodes: each AND node is identified by a number, say Z. Associated with each AND node is a string 'A' Z which contains information on that node and its function in the tree. The content of 'A' Z will vary during execution as follows:

1) if node Z is pruned from the tree then

$('A' Z) is null

2) if node Z is TRUE then

$('A' Z) = 'I' IN 'UT' LIST

where IN = the associated input symbol and

LIST = the list of TRUE successor OR nodes (in the form #,#,...,#,)

3) if node Z is of undetermined truth value

$('A' Z) = 'I' IN 'PR' X 'U' LIST1 'T' LIST2

where IN = the associated input symbol

X = the number of the immediate OR predecessor

LIST1 = the list of unvalued immediate successor OR nodes and

LIST2 = the list of TRUE immediate successor OR nodes

AD = the list of AND nodes that should be pruned from the tree, in the form #,#,...,#,

AM = the list of admissible states, in the form #,#,...,#,

$('FS' S '.' I) = FS(S,I)

$('FZ' S '.' I) = FZ(S,I)

NAME = the name of M

NAND = the number of the next AND node to be created

NOR = the number of the next OR node to be created

OA = the list of the nonterminal OR nodes with as yet uncomputed successors, in the form #,#,...,#,

OD = the list of OR nodes that should be pruned from the tree, in the form #,#,...,#,

During the construction and testing of all I/J successors of an unvalued OR node the following temporary strings are created.

OAT = the list of information on nonterminal OR node successors to a new AND node, each entry in the list is of the form

NO 'L' LG 'PT' SET 'O' OUT 'PR' Z 'S'

where

NO = the number of the new OR node and the remainder of the entry is as previously defined for a nonterminal OR node.

OS = the list of new AND node successors

SSW = the program switch which indicates the truth value of the OR node whose successors are being created

SW = the program switch which indicates if a successful experiment has been found

TT = the list of information on TRUE terminal OR node successors to a new AND node, each entry in the list is of the form

NO 'F' FIN 'I' INIT 'O' OUT '/'

where

NO = the number of the new OR node and the remainder of the entry is as previously defined for a terminal OR node.

Y = the current I/J successor set being

examined, in the form #. #,#. #,. . . ,#. #,

J. Annotated Flow Chart

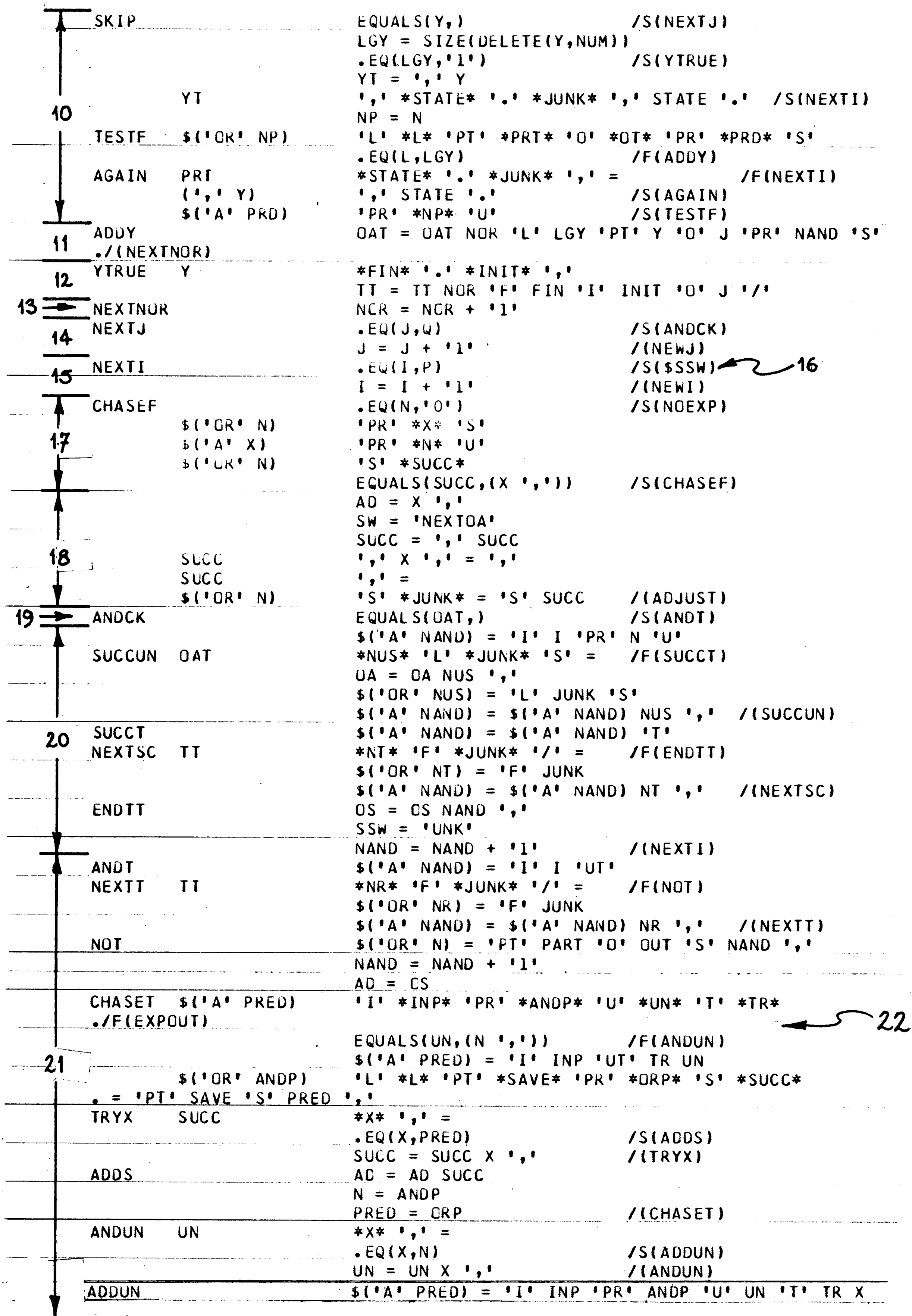
START

1. INPUT NAME, N, P, Q, FS AND FZ-TABLES; PRINT NAME, N, P, Q, FS AND FZ-TABLES
2. INPUT AM; PRINT AM
3. |AM| = 1 ?
4. LOAD 'OR' O, ADD O TO OA
5. LET N = 1ST TERM ON OA, DELETE
6. DOES 'OR' N EXIST ?
7. OS = SSW = 'CHASEF'
8. TT = OAT =
9. Y = I/J SUCC OF 'OR' N
10. BRANCH ON Y (TRUE / UNKNOWN / FALSE / φ)
11. ADD Y TO OAT
12. ADD Y TO TT
13. INCREMENT NOR
14. J > Q ? (J=1, J=J+1)
15. I > P ? (I=1, I=I+1)
16. BRANCH ON SSW (UNK / CHASEF)
17. CHASE FALSE BACK TOWARD ROOT NODE; LET N = HIGHEST FALSE OR NODE, X = HIGHEST FALSE AND NODE
18. ADD X TO AD, SET SW = 'NEXTOA', REVISE SUCC LIST OF 'OR' N
19. IS AND NODE TRUE ?
20. CREATE 'A' NAND; ADD OAT TO OA; CREATE NEW OR NODES FROM OAT, DELETE; CREATE NEW OR NODES FROM TT; ADD NAND TO OS; SET SW = 'UNK'; INCREMENT NAND
21. CREATE 'A' NAND; CREATE NEW OR NODES FROM TT; ADD AND AS SUCC OF 'OR' N; INCREMENT NAND; SET AD = OS; CHASE TRUE BACK TOWARD ROOT NODE ADJUSTING OR NODE SUCC LISTS AND ADDING TO AD
22. 'OR' O = TRUE ?
23. SW = 'NEXTOA'
24. SW = 'EXP'
25. AD = φ ?
26. LET NO = 1ST TERM ON AD, DELETE; ADD SUCC'S OF 'A' NO TO OD; DELETE 'A' NO
27. OD = φ ?
28. LET NO = 1ST TERM ON OD, DELETE; ADD SUCC'S OF 'OR' NO TO AD; DELETE 'OR' NO
29. AD = φ ?
30. BRANCH ON SW (EXP / NEXTOA)
31. ADD OS TO 'OR' N
32. PRINT 'NO EXPERIMENT'
33. IS ANOTHER AM ON SYSPIT ?
34. PRINT TITLES; SET OT = O
35. OT = φ ?
36. LET N = 1ST TERM ON 'OT', DELETE; B (= SUCC OF 'OR' N) ⇒ INPUT I; PRINT 'OR' N, I; ADD SUCC OF B TO OT, ORDER SUCC'S BY OUTPUTS AND PRINT
37. PRINT 'AM = SINGLETON'

K. Annotated Program Listing

SNOBOL (31 DEC 1965 VERSION) PROGRAM LISTING ... ...

```
START                   NUM = '0123456789.'
NEXTM                   NAME = TRIM(SYSPIT)
                        SYSPOT = '1SIMPLE ADAPTIVE DIAGNOSING'
                        SYSPOT = ' '
                        SYSPOT = 'MACHINE NAME = ' NAME
                        SYSPOT = ' '
          TRIM(SYSPIT)  *N* ',' *P* ',' *Q*
                        SYSPOT = 'N = ' N ', P = ' P ', Q = ' Q
                        SYSPOT = ' '
                        SYSPOT = 'TRANSITION TABLES***STATE FOLLOWED '
 . 'BY LIST OF (NEXT STATE/OUTPUT) FOR SUCCESSIVE INPUTS'
                        SYSPOT = ' '
NEXTX     TRIM(SYSPIT)  *X* ',' *LIST*
                        SYSPOT = ' ' X '     ' LIST
                        LIST = LIST ','
                        Y = '0'
NEXTY                   Y = Y + '1'
          LIST          *STATE* '/' *OUTPUT* ',' =
                        $('FS' X '.' Y) = STATE
                        $('FZ' X '.' Y) = OUTPUT
                        .EQ(Y,P)                 /F(NEXTY)
                        .EQ(X,N)                 /F(NEXTX)
NEXTAM    SYSPIT        '(' *AM* ')'
                        SYSPOT = '1MACHINE NAME = ' NAME
 . ', ADMISSIBLE SET = (' AM ')'
                        SYSPOT = ' '
                        .EQ('0',SIZE(DELETE(AM,NUM)))      /S(SINGLE)
                        AM = AM ','
                        ORO = 'L' SIZE(DELETE(AM,NUM)) 'PT'
MORE      AM            *STATE* ',' =            /F(NOMORE)
                        ORO = ORO STATE '.' STATE ','      /(MORE)
NOMORE                  ORO = ORO 'OPRS'
                        OA = '0,'
                        NCR = '1'
                        NAND = '1'
NEXTOA    OA            *N* ',' =
          $('OR' N)     'L' *LG* 'PT' *PART* 'O' *OUT* 'PR' *PRED* 'S'
 . /F(NEXTOA)
                        I = '1'
                        OS =
                        SSW = 'CHASEF'
NEWI                    TT =
                        OAT =
                        J = '1'
NEWJ                    Y =
                        PT = PART
NEXTP     PT            *STATE* '.' *INIT* ',' =          /F(SKIP)
                        .NE(J,$('FZ' STATE '.' I))        /S(NEXTP)
                        Y = Y $('FS' STATE '.' I) '.' INIT ',' /(NEXTP)
```



```
SKIP                      EQUALS(Y,)                     /S(NEXTJ)
                          LGY = SIZE(DELETE(Y,NUM))
                          .EQ(LGY,'1')                   /S(YTRUE)
                          YT = ',' Y
          YT              ',' *STATE* '.' *JUNK* ',' STATE '.'   /S(NEXTI)
                          NP = N
TESTF     $('OR' NP)      'L' *L* 'PT' *PRT* 'O' *OT* 'PR' *PRD* 'S'
                          .EQ(L,LGY)                     /F(ADDY)
AGAIN     PRT             *STATE* '.' *JUNK* ',' =               /F(NEXTI)
          (',' Y)         ',' STATE '.'                  /S(AGAIN)
          $('A' PRD)      'PR' *NP* 'U'                  /S(TESTF)
ADDY                      OAT = OAT NOR 'L' LGY 'PT' Y 'O' J 'PR' NAND 'S'
./(NEXTNOR)
YTRUE     Y               *FIN* '.' *INIT* ','
                          TT = TT NOR 'F' FIN 'I' INIT 'O' J '/'
NEXTNOR                   NOR = NOR + '1'
NEXTJ                     .EQ(J,Q)                       /S(ANDCK)
                          J = J + '1'                    /(NEWJ)
NEXTI                     .EQ(I,P)                       /S($SSW)
                          I = I + '1'                    /(NEWI)
CHASEF                    .EQ(N,'0')                     /S(NOEXP)
          $('OR' N)       'PR' *X* 'S'
          $('A' X)        'PR' *N* 'U'
          $('OR' N)       'S' *SUCC*
                          EQUALS(SUCC,(X ','))           /S(CHASEF)
                          AD = X ','
                          SW = 'NEXTOA'
                          SUCC = ',' SUCC
          SUCC            ',' X ',' = ','
          SUCC            ',' =
          $('OR' N)       'S' *JUNK* = 'S' SUCC          /(ADJUST)
ANDCK                     EQUALS(OAT,)                   /S(ANDT)
                          $('A' NAND) = 'I' I 'PR' N 'U'
SUCCUN    OAT             *NUS* 'L' *JUNK* 'S' =         /F(SUCCT)
                          OA = OA NUS ','
                          $('OR' NUS) = 'L' JUNK 'S'
                          $('A' NAND) = $('A' NAND) NUS ','   /(SUCCUN)
SUCCT                     $('A' NAND) = $('A' NAND) 'T'
NEXTSC    TT              *NT* 'F' *JUNK* '/' =          /F(ENDTT)
                          $('OR' NT) = 'F' JUNK
                          $('A' NAND) = $('A' NAND) NT ','    /(NEXTSC)
ENDTT                     OS = OS NAND ','
                          SSW = 'UNK'
                          NAND = NAND + '1'              /(NEXTI)
ANDT                      $('A' NAND) = 'I' I 'UT'
NEXTT     TT              *NR* 'F' *JUNK* '/' =          /F(NOT)
                          $('OR' NR) = 'F' JUNK
                          $('A' NAND) = $('A' NAND) NR ','    /(NEXTT)
NOT                       $('OR' N) = 'PT' PART 'O' OUT 'S' NAND ','
                          NAND = NAND + '1'
                          AD = CS
CHASET    $('A' PRED)     'I' *INP* 'PR' *ANDP* 'U' *UN* 'T' *TR*
./F(EXPOUT)
                          EQUALS(UN,(N ','))             /F(ANDUN)
                          $('A' PRED) = 'I' INP 'UT' TR UN
          $('OR' ANDP)    'L' *L* 'PT' *SAVE* 'PR' *ORP* 'S' *SUCC*
. = 'PT' SAVE 'S' PRED ','
TRYX      SUCC            *X* ',' =
                          .EQ(X,PRED)                    /S(ADDS)
                          SUCC = SUCC X ','              /(TRYX)
ADDS                      AD = AD SUCC
                          N = ANDP
                          PRED = ORP                     /(CHASET)
ANDUN     UN              *X* ',' =
                          .EQ(X,N)                       /S(ADDUN)
                          UN = UN X ','                  /(ANDUN)
ADDUN                     $('A' PRED) = 'I' INP 'PR' ANDP 'U' UN 'T' TR X
```

```
23                                  SW = 'NEXTOA'              /(ADJUST)
24      EXPOUT                      SW = 'EXP'
25      ADJUST   AD                 *NO* ',' =                 /F(ORCK)
                 $('A' NO)          *JUNK* 'U' *UN* 'T' *TR* =
26                                  OC = OD UN TR              /(ADJUST)
27      ORCK     OC                 *NO* ',' =                 /F(FINCK)
                 $('OR' NO)         *JUNK* 'S' *SUCC* =        /F(ORT)        30
28                                  AD = AD SUCC               /(ORCK)
        ORT                         $('OR' NO) =               /(ORCK)
29      FINCK                       EQUALS(AD,)                /S($SW)F(ADJUST)
        UNK                         $('OR' N) = 'L' LG 'PT' PART 'O' OUT 'PR' PRED
31      . 'S' OS         /(NEXTOA)
        NOEXP                       SYSPOT = ' '
32                                  SYSPOT = 'NO SIMPLE ADAPTIVE DIAGNOSING '
        . 'EXPERIMENT EXISTS FOR THE GIVEN MACHINE AND ADMISSIBLE SET'
33      LOKAM    SYSLCK             '(' *AM* ')'               /S(NEXTAM)F(NEXTM)
        EXP                         SYSPOT = ' '
                                    SYSPOT = 'THE FOLLOWING IS A SIMPLE ADAPTIVE '
        . 'DIAGNOSING EXPERIMENT FOR THE GIVEN MACHINE AND ADMISSIBLE SET'
34                                  SYSPOT = ' '
                                    SYSPOT = 'NO    SET     INPUT, FOLLOWED BY'
        . ' THE (OUTPUT/NEXT NO) LIST'
                                    OT = '0,'
35      NEXTOR   OT                 *N* ',' =                  /F(LOKAM)
                 $('OR' N)          'PT' *PART* ',O' *OUT* 'S' *SUCC* ',' =
                 $('A' SUCC)        'I' *IN* 'UT' *TR* =
                                    SYSPOT = ' '
                                    SYSPOT = ' ' N '    ' PART '    ' IN
        NEXTS    TR                 *X* ',' =                  /F(OUTPUT)
                 $('OR' X)          'C' *OUT* 'S'              /F(TRUE)
                                    $('S' OUT) = X             /(NEXTS)
        TRUE     $('OR' X)          'F' *FIN* 'I' *INIT* 'O' *OUT* =
36                                  $('S' OUT) =  'STOP(INIT = ' INIT ',FIN = '
        . FIN ')'        /(NEXTS)
        OUTPUT                      N = '1'
        NEWN                        EQUALS($('S' N),)          /S(NEXTN)
                                    SYSPOT = '          ' N '/' $('S' N)
                 $('S' N)           ANCHOR() 'S'               /S(ERASE)
                                    OT = OT $('S' N) ','
        ERASE                       $('S' N) =
        NEXTN                       .EQ(N,Q)                   /S(NEXTOR)
                                    N = N + '1'                /(NEWN)
37      SINGLE                      SYSPOT = 'ADMISSIBLE SET IS A SINGLETON***'
        . 'NO EXPERIMENT NECESSARY'                            /(LOKAM)
        END

  SUCCESSFUL COMPILATION
```

IV. Finding an Equivalent Regular Expression

A. Program name: RE

B. Purpose: To display a regular expression equivalent to the set of input strings accepted by a given Moore or Mealy machine with initial state and accepting output specified.

C. Method: Given any finite-state machine one can compute the regular expression α_{ij}^{k}, which denotes all input strings that take the given machine from state i to state j without passing through any state with index > k, via the recursion equation

$$\alpha_{ij}^{k} = \alpha_{ij}^{k-1} \cup \alpha_{ik}^{k-1} (\alpha_{kk}^{k-1})^{*} \alpha_{kj}^{k-1}$$

and terminal equations

$$\alpha^{0} = \Lambda \cup \{x \mid FS(i,x) = i\}$$

where Λ is the null string (ϕ = null set)

and

$$\alpha_{ij}^{0} = \{x \mid FS(i,x) = j\} \qquad i \neq j.$$

Notes:

1) Contrary to McNaughton-Yamada α_{ii}^{0} can never be the empty set (See Harrison [3], p. 325).

2) α_{ij}^{k} is a function only of the set of α's for k-1; thus the k-1 set of α's can be discarded once the k set is known.

The regular expression equivalent to a given Moore machine with initial state i and accepting output z is

$$\bigcup_{FZ(j)=z} \alpha_{ij}^{N} ;$$

and the regular expression equivalent to a given Mealy machine with initial state i and accepting output z is

$$\bigcup_{j \in S} \alpha_{ij}^{N} \beta_{jz}$$

where

$$\beta_{jz} = \{x \mid FZ(j,x) = z\}.$$

In the program being described the recursion equation for evaluating α_{ij}^{k} is not implemented blindly; rather, attempts are made to identify and exploit simple identities - such as

$$\alpha \cup \phi = \phi \cup \alpha = \alpha$$

$$\alpha^* \cup \Lambda = \Lambda \cup \alpha^* = \alpha^*$$

$$\vdots$$

etc.

To assist in this endeavor the program attaches to each α-string a type code selected according to the following scheme (every α-string will be in the form C ',' R where C is the type code and R is a regular expression):

C	Implied Regular Expression
PL	ϕ (the empty set); R is blank
L	Λ (the null tape); R is blank
1	R (R = singleton $\neq \Lambda$)
E	$\Lambda \cup R$ ($\Lambda \notin R$ = single string of concatenated RE's)
I	R ($\Lambda \in R$ = single string of concatenated RE's)
S	R ($\Lambda \notin R$ = union of several RE's)
1E	$\Lambda \cup R$ ($\Lambda \notin R$ = singleton)
SE	$\Lambda \cup R$ ($\Lambda \notin R$ = union of several RE's)
SI	R ($\Lambda \in R$ = union of several RE's)
I*	R* (R = single string of concatenated RE's)
1I*	R* (R = singleton $\neq \Lambda$)
SI*	R* (R = union of several RE's)
blank	R ($\Lambda \notin R$ = single string of concatenated RE's)

Not all simplifications are detected, however, and it is frequently the case that the observant user can further simplify the results this program produces.

D. Language and System: The program is written in SNOBOL (31 December 1965 version) for execution on the University of Michigan Executive System for the IBM 7090 Computer as it existed in May 1967.

E. Input: The input deck consists of an arbitrary number of sub-decks; each sub-deck names and describes a finite-state machine and lists one or more pairs of initial state and accepting output relative to which equivalent regular expressions are sought. The make-up of each sub-deck and the card formats are as follows:

1) Name card - up to 80 columns are available for any string of alphameric characters identifying the machine by name; trailing blanks are ignored. This card must head each sub-deck even if the name is blank.

2) N, P, Q, T card - gives the values of N, P, Q and specifies the type of the machine; three integers and a code word separated by commas, no blanks, left justified on the card. The code word shall be MOORE or MEALY according to the type of machine being described.

3) Transition table cards N cards, each of the following form:

   $s/j,\ s_1, s_2, \ldots, s_P$ for a Moore machine

   and

   $s,\ s_1/j_1,\ s_2/j_2,\ \ldots,\ s_P/j_P$ for a Mealy machine

   where all data is left justified with no blanks and where $s \epsilon S$

$j = FZ(s)$ for a Moore machine

$s_i = FS(s, i)$

$j_i = FZ(s, i)$ for a Mealy machine.

There should be precisely one transition table card of the appropriate type for each state in M; the transition table cards may be in any order provided the last card has $s = N$.

4) Initial state/accepting output card - a single card specifying one or more initial state-accepting output pairs relative to which equivalent regular expressions on the given machine are sought. Data should appear in the following form:

$s_1/z_1, s_2/z_2, \ldots, s_k/z_k$

where $s_i \in S$, $z_i \in A_o$ and where all data is left justified with no blanks.

Restrictions: The only limitations on the size of machines (i. e., N, P, Q) or number of initial state-accepting output pairs this program will handle are those imposed by the SNOBOL compiler itself on string length, number of strings, etc. .... However, the input routines for this program expect to find each of the following data entries on a single card:

1) the machine name

2) the N, P, Q, T data

3) each row of the transition table

4) the set of initial state-accepting output pairs.

If multiple card input is required the input routines can be rewritten; no change should be necessary in the body of the program.

F. Operation and Output: The program works through the data deck sub-deck by sub-deck and for each will output a description of the corresponding machine followed by a regular expression for each initial state-accepting output pair. The letter L is used to denote the null string in the output regular expressions.

G. Sample Data

```
M DEMO
9,2,2,MOORE
1/1,1,1
2/1,3,3
3/2,3,3
4/2,1,1
5/2,1,6
6/1,5,5
7/2,8,1
8/1,9,1
9/2,1,9
1/2,4/2,7/2,9/2,5/2,8/2
```

3/2

3/1,2,1

2/1,2,3

1/2,3,2

3,2,2,MOORE

MC NAUGHTON, YAMADA FIG. 4

2/2,3/1

3/1,3,2

2/1,3,1

1/2,2,3

3,2,2,MOORE

MC NAUGHTON, YAMADA FIG. 3

1/2,2/2,7/1

7,6/2,7/2,7/2

6,7/1,6/1,6/1

5,7/2,4/1,4/1

4,6/2,5/1,5/1

3,3/2,5/1,3/1

2,2/1,5/1,2/1

1,1/1,4/1,1/1

7,3,2,MEALY

41

H. Sample Output

```
EQUIVALENT REGULAR EXPRESSION EVALUATION

MACHINE NAME = M DEMO
N = 9, P = 2, Q = 2, TYPE = MOORE

TRANSITION TABLES***STATE/CUTPUT FOLLOWED BY LIST OF NEXT STATES FCR SUCCESSIVE INPUTS

 1/1          1,1
 2/1          3,3
 3/2          3,3
 4/2          1,1
 5/2          1,6
 6/1          5,5
 7/2          8,1
 8/1          9,1
 9/2          1,9

 MACHINE NAME = M DEMO, INITIAL STATE = 1, ACCEPTING CUTPUT = 2

EQUIVALENT RE = EMPTY

 MACHINE NAME = M DEMO, INITIAL STATE = 4, ACCEPTING OUTPUT = 2

EQUIVALENT RE = L

 MACHINE NAME = M DEMO, INITIAL STATE = 7, ACCEPTING OUTPUT = 2

EQUIVALENT RE = LU11U112*

 MACHINE NAME = M DEMO, INITIAL STATE = 9, ACCEPTING OUTPUT = 2

EQUIVALENT RE = 2*

 MACHINE NAME = M DEMO, INITIAL STATE = 5, ACCEPTING OUTPUT = 2

EQUIVALENT RE = LU2((1U2)2)*(1U2)

 MACHINE NAME = M DEMO, INITIAL STATE = 8, ACCEPTING OUTPUT = 2

EQUIVALENT RE = 1U12*
```

```
EQUIVALENT REGULAR EXPRESSION EVALUATION

MACHINE NAME = M1
N = 7, P = 3, Q = 2, TYPE = MEALY

TRANSITION TABLES***STATE FOLLOWED BY LIST OF (NEXT STATE/OUTPUT) FOR SUCCESSIVE INPUTS

  1          1/1,4/1,1/1
  2          2/1,5/1,2/1
  3          3/2,5/1,3/1
  4          6/2,5/1,5/1
  5          7/2,4/1,4/1
  6          7/1,6/1,6/1
  7          6/2,7/2,7/2

  MACHINE NAME = M1, INITIAL STATE = 1, ACCEPTING OUTPUT = 2

 EQUIVALENT RE = (2U(1U3)*2U(2U(1U3)*2)(2U3)((2U3)(2U3))*(2U3))1U((2U(1U3)*2)(2U3)U(2U(1U3)*2)(2U3)((2U3)(2U3))*)1U((2U(1U3)*2)(2U3
                 )((2U3)(2U3))*1U((2U(1U3)*2)1U(2U(1U3)*2)(2U3)((2U3)(2U3))*(2U3)1)(2U3)*1U((2U(1U3)*2)(2U3)((2U3)(2U3))*1U((2U
                 (1U3)*2)1U(2U(1U3)*2)(2U3)((2U3)(2U3))*(2U3)1)(2U3)*1)(2U3U1(2U3)*1)*)(1U2U3)

  MACHINE NAME = M1, INITIAL STATE = 2, ACCEPTING OUTPUT = 2

 EQUIVALENT RE = (2U(1U3)*2)((2U3)(2U3))*(2U3)1U(2U(1U3)*2U(2U(1U3)*2)((2U3)(2U3))*)1U((2U(1U3)*2)((2U3)(2U3))*1U(2U(1U3)*2)((2U3)(
                 2U3))*(2U3)1(2U3)*1U((2U(1U3)*2)((2U3)(2U3))*1U(2U(1U3)*2)((2U3)(2U3))*(2U3)1(2U3)*1)(2U3U1(2U3)*1)*)(1U2U3)

  MACHINE NAME = M1, INITIAL STATE = 7, ACCEPTING OUTPUT = 1

 EQUIVALENT RE = (1U1(2U3)*U(2U3U1(2U3)*1)*(1U1(2U3)*))(1U2U3)
```

```
  EQUIVALENT REGULAR EXPRESSION EVALUATION

 MACHINE NAME = MC NAUGHTON, YAMADA FIG. 3
 N = 3, P = 2, Q = 2, TYPE = MOORE

 TRANSITION TABLES***STATE/OUTPUT FOLLOWED BY LIST OF NEXT STATES FOR SUCCESSIVE INPUTS

  1/2          2,3
  2/1          3,1
  3/1          3,2

  MACHINE NAME = MC NAUGHTON, YAMADA FIG. 3, INITIAL STATE = 2, ACCEPTING OUTPUT = 2

 EQUIVALENT RE = 2U(21)*2U(1U22U(21)*(1U22))(1U2(21)*(1U22))*2(21)*2

  MACHINE NAME = MC NAUGHTON, YAMADA FIG. 3, INITIAL STATE = 3, ACCEPTING OUTPUT = 1

 EQUIVALENT RE = 2U2(21)*U(1U2(21)*(1U22))*(2U2(21)*)U(1U2(21)*(1U22))*
```

```
EQUIVALENT REGULAR EXPRESSION EVALUATION

MACHINE NAME = MC NAUGHTON, YAMADA FIG. 4
N = 3, P = 2, Q = 2, TYPE = MOORE

TRANSITION TABLES***STATE/OUTPUT FOLLOWED BY LIST OF NEXT STATES FOR SUCCESSIVE INPUTS

 1/2          3,2
 2/1          2,3
 3/1          2,1

MACHINE NAME = MC NAUGHTON, YAMADA FIG. 4, INITIAL STATE = 3, ACCEPTING OUTPUT = 2

EQUIVALENT RE = 2U(21U(1U22)1*2)*2

**** ALL INPUT DATA HAVE BEEN PROCESSED

          ELAPSED TIME --

          TOTAL              PROCESSING          EXECUTION          LOADING

           5' 59.1"           0'  1.9"            5' 51.0"           0'  6.2"
```

I. Important Program Variables: This portion of the report will use the conventions of SNOBOL to describe the contents of several important string variables appearing in the program.

$('A' I '.' J '.' X) = a string of the form C '.' R where C is a type code and R a regular expression; the set of input strings denoted by C and R is α^{k}_{IJ}.

$('A' I '.' J '.' Y) = a string of the same form as above which denotes α^{k-1}_{IJ} .

$('FY' S '.' Z) = $\beta_{SZ} = \{x \mid FZ(S,x) = Z\}$, for Mealy machines only.

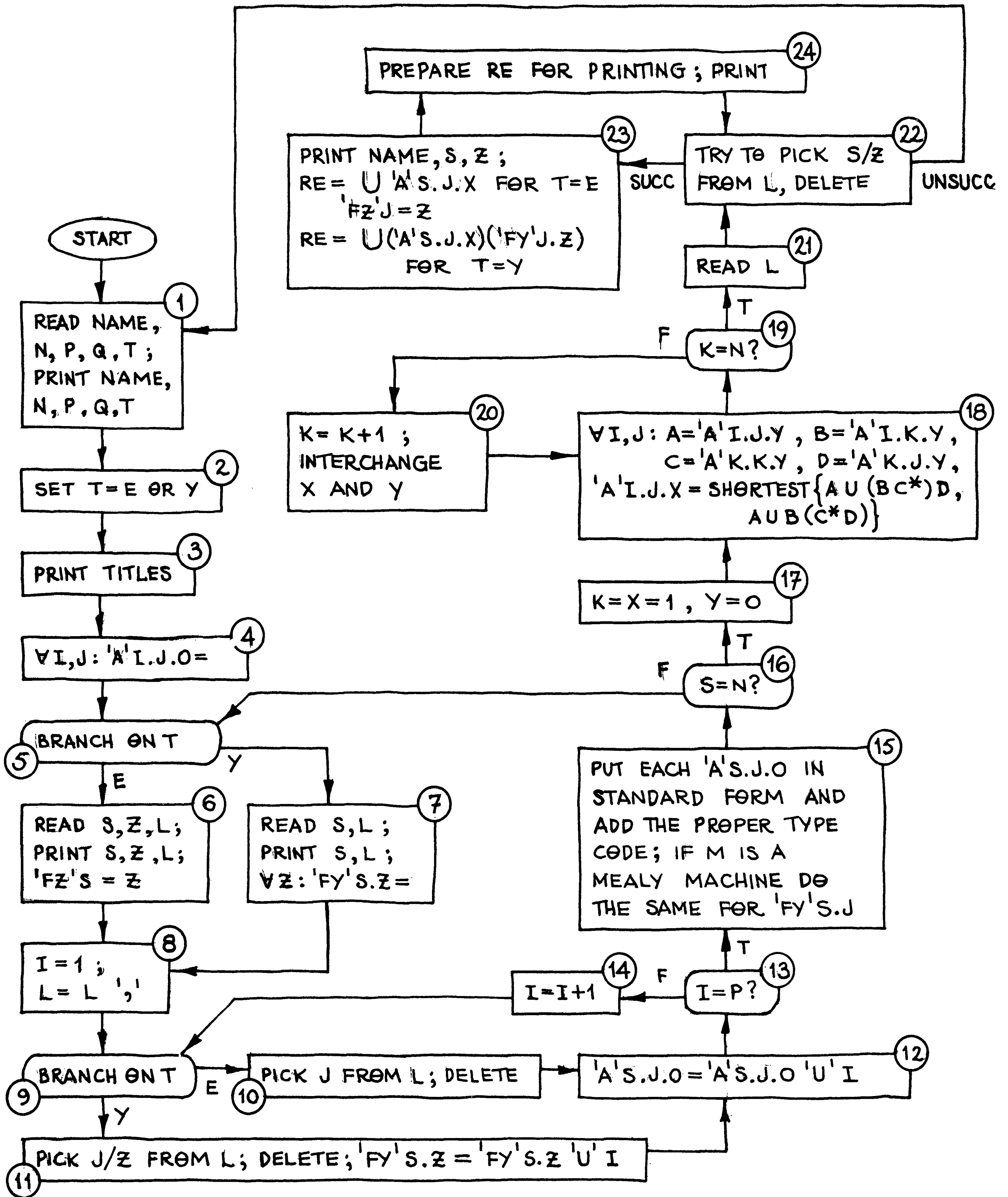
$('FZ' S) = FZ(S), for Moore machines only.

PROD = a subroutine which concatenates two given regular expressions; i. e., PROD(A, B) = AB.

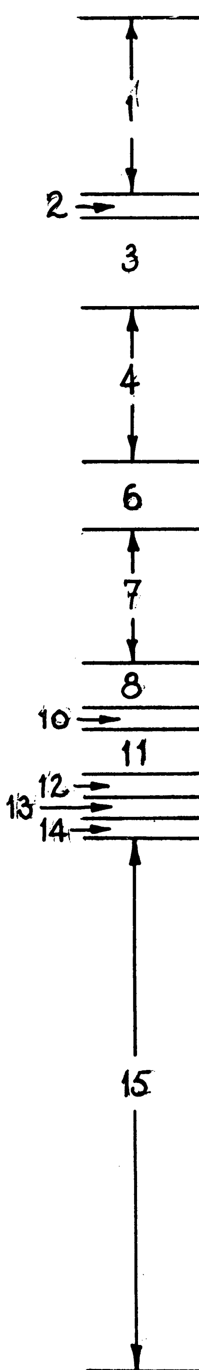
STAR = a subroutine which stars a given regular expression; i. e., STAR(A) = A*.

UNION a subroutine which yields the union of two given regular expressions, i. e. UNION(A, B) = $A \cup B$.

J. Annotated Flow Chart

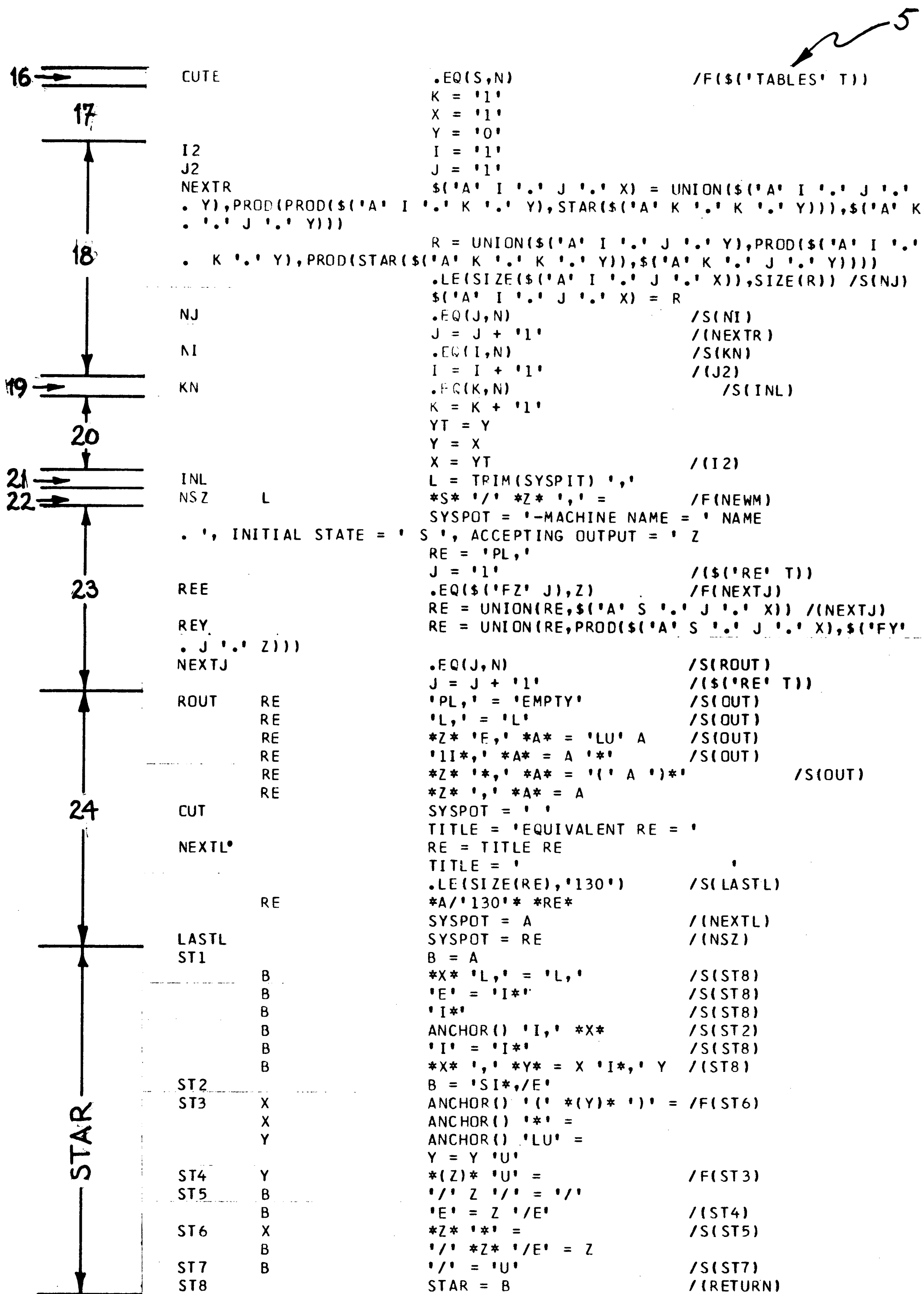


K. Annotated Program Listing




```
SNOBOL (31 DEC 1965 VERSION) PROGRAM LISTING ... ... ...

 DEFINE('STAR(A)' , 'ST1' , 'B,X,Y,Z')
 DEFINE('UNION(A,B)' , 'UN1' , 'C,TA,TB,TAB,SA,SB,X,YA,YB,Z,ZA,ZB')
 DEFINE('PROD(A,B)' , 'PR1' , 'C,TA,TB,SA,SB,TAB,X,Y')
START                     AE = '/OUTPUT'
                          AY =
                          BE = 'NEXT STATES'
                          BY = '(NEXT STATE/OUTPUT)'
NEWM                      NAME = TRIM(SYSPIT)
        TRIM(SYSPIT)      *N* ',' *P* ',' *Q* ',' *T*
                          SYSPOT = '1EQUIVALENT REGULAR EXPRESSION'
. ' EVALUATION'
                          SYSPOT = ' '
                          SYSPOT = 'MACHINE NAME = ' NAME
                          SYSPOT = 'N = ' N ', P = ' P ', Q = ' Q
. ', TYPE = ' T
        T                 *X/'4'* =
                          SYSPOT = ' '
                          SYSPOT = 'TRANSITION TABLES***STATE' $('A' T)
.' FOLLOWED BY LIST OF ' $('B' T) ' FOR SUCCESSIVE INPUTS'
                          SYSPOT = ' '
                          I = '1'
J1                        J = '1'
NULL                      $('A' I '.' J '.0') =
                          .EQ(J,N)                  /S(TESTI)
                          J = J + '1'               /(NULL)
TESTI                     .EQ(I,N)                  /S($('TABLES' T))
                          I = I + '1'               /(J1)
TABLESE TRIM(SYSPIT)      *S* '/' *Z* ',' *L*
                          SYSPOT = ' ' S '/' Z '          ' L
                          $('FZ' S) = Z             /(I1)
TABLESY TRIM(SYSPIT)      *S* ',' *L*
                          SYSPOT = ' ' S '          ' L
                          I = '1'
NEXTF1                    $('FY' S '.' I) =
                          .EQ(I,Q)                  /S(I1)
                          I = I + '1'               /(NEXTF1)
I1                        I = '1'
                          L = L ','                 /($('L' T))
LE       L                *J* ',' =                 /(ADDA)
LY       L                *J* '/' *Z* ',' =
                          $('FY' S '.' Z) = $('FY' S '.' Z) 'U' I
ADDA                      $('A' S '.' J '.0') = $('A' S '.' J '.0') 'U' I
                          .EQ(I,P)                  /S(CLEAN)
                          I = I + '1'               /($('L' T))
CLEAN                     J = '1'
EMPCK                     EQUALS($('A' S '.' J '.0'),)       /S(EMP)
        $('A' S '.' J '.0')  'U' =
                          .EQ(SIZE($('A' S '.' J '.0')),'1') /S(SING)
                          X = 'S'                   /(ETEST)
SING                      X = '1'
ETEST                     .EQ(S,J)                  /F(FIN)
                          X = X 'E'
FIN                       $('A' S '.' J '.0') = X ',' $('A' S '.' J '.0')
. /(CKJ)
EMP                       .EQ(S,J)                  /S(L)
                          $('A' S '.' J '.0') = 'PL,'        /(CKJ)
L                         $('A' S '.' J '.0') = 'L,'
CKJ                       .EQ(J,N)                  /S($('OUT' T))
                          J = J + '1'               /(EMPCK)
OUTY                      J = '1'
EMPCK1                    EQUALS($('FY' S '.' J),) /S(EMP1)
        $('FY' S '.' J) 'U' =
                          .EQ(SIZE($('FY' S '.' J)),'1')     /S(SING1)
                          $('FY' S '.' J) = 'S,' $('FY' S '.' J)  /(CKJ1)
SING1                     $('FY' S '.' J) = '1,' $('FY' S '.' J) /(CKJ1)
EMP1                      $('FY' S '.' J) = 'PL,'
CKJ1                      .EQ(J,Q)                  /S(OUTE)
                          J = J + '1'               /(EMPCK1)
```




```
CUTE                          .EQ(S,N)                /F($('TABLES' T))
                              K = '1'
                              X = '1'
                              Y = '0'
I2                            I = '1'
J2                            J = '1'
NEXTR                         $('A' I '.' J '.' X) = UNION($('A' I '.' J '.'
. Y),PROD(PROD($('A' I '.' K '.' Y),STAR($('A' K '.' K '.' Y))),$('A' K
. '.' J '.' Y)))
                              R = UNION($('A' I '.' J '.' Y),PROD($('A' I '.'
.  K '.' Y),PROD(STAR($('A' K '.' K '.' Y)),$('A' K '.' J '.' Y))))
                              .LE(SIZE($('A' I '.' J '.' X)),SIZE(R)) /S(NJ)
                              $('A' I '.' J '.' X) = R
NJ                            .EQ(J,N)                /S(NI)
                              J = J + '1'             /(NEXTR)
NI                            .EQ(I,N)                /S(KN)
                              I = I + '1'             /(J2)
KN                            .EQ(K,N)                   /S(INL)
                              K = K + '1'
                              YT = Y
                              Y = X
                              X = YT                  /(I2)
INL                           L = TRIM(SYSPIT) ','
NSZ         L                 *S* '/' *Z* ',' =       /F(NEWM)
                              SYSPOT = '-MACHINE NAME = ' NAME
. ', INITIAL STATE = ' S ', ACCEPTING OUTPUT = ' Z
                              RE = 'PL,'
                              J = '1'                 /($('RE' T))
REE                           .EQ($('FZ' J),Z)        /F(NEXTJ)
                              RE = UNION(RE,$('A' S '.' J '.' X)) /(NEXTJ)
REY                           RE = UNION(RE,PROD($('A' S '.' J '.' X),$('FY'
. J '.' Z)))
NEXTJ                         .EQ(J,N)                /S(ROUT)
                              J = J + '1'             /($('RE' T))
ROUT        RE                'PL,' = 'EMPTY'         /S(OUT)
            RE                'L,' = 'L'              /S(OUT)
            RE                *Z* 'E,' *A* = 'LU' A   /S(OUT)
            RE                '1I*,' *A* = A '*'      /S(OUT)
            RE                *Z* '*,' *A* = '(' A ')*'          /S(OUT)
            RE                *Z* ',' *A* = A
CUT                           SYSPOT = ' '
                              TITLE = 'EQUIVALENT RE = '
NEXTL                         RE = TITLE RE
                              TITLE = '                    '
                              .LE(SIZE(RE),'130')     /S(LASTL)
            RE                *A/'130'* *RE*
                              SYSPOT = A              /(NEXTL)
LASTL                         SYSPOT = RE             /(NSZ)
ST1                           B = A
            B                 *X* 'L,' = 'L,'         /S(ST8)
            B                 'E' = 'I*'              /S(ST8)
            B                 'I*'                    /S(ST8)
            B                 ANCHOR() 'I,' *X*       /S(ST2)
            B                 'I' = 'I*'              /S(ST8)
            B                 *X* ',' *Y* = X 'I*,' Y /(ST8)
ST2                           B = 'SI*,/E'
ST3         X                 ANCHOR() '(' *(Y)* ')' = /F(ST6)
            X                 ANCHOR() '*' =
            Y                 ANCHOR() 'LU' =
                              Y = Y 'U'
ST4         Y                 *(Z)* 'U' =             /F(ST3)
ST5         B                 '/' Z '/' = '/'
            B                 'E' = Z '/E'            /(ST4)
ST6         X                 *Z* '*' =               /S(ST5)
            B                 '/' *Z* '/E' = Z
ST7         B                 '/' = 'U'               /S(ST7)
ST8                           STAR = B                /(RETURN)
```

```
PROD

PR1     A           *TA* ',' *SA*
        B           *TB* ',' *SB*
                    TAB = TA TB
        TAB         'P'                          /S(PR10)
        TAB         'L' =                        /S(PR11)
                    EQUALS(SA,SB)                /F(PR13)
        TAB         '*' *X* 'I'                  /S(PR14)
        TAB         '*' *X* 'E'                  /F(PR2)
PR14                C = A                        /(PR12)
PR2     (TB TA)     '*' *X* 'I'                  /S(PR15)
        (TB TA)     '*' *X* 'E'                  /F(PR13)
PR15                C = B                        /(PR12)
PR3                 X = 'A'
PR4                 Y = $('T' X)
        Y           'E'                          /F(PR5)
                    $('S' X) = '(LU' $('S' X) ')'          /(PR7)
PR5     Y           'S'                          /S(PR6)
        Y           ANCHOR() 'I*'                /F(PR7)
PR6                 $('S' X) = '(' $('S' X) ')'
PR7     $('T' X)    'E' = 'I'
        Y           '*'                          /F(PR8)
                    $('S' X) = $('S' X) '*'
PR8     X           'A' = 'B'                    /S(PR4)
        (TA TB)     'I' *X* 'I'                  /S(PR9)
                    C = ',' SA SB                /(PR12)
PR9                 C = 'I,' SA SB               /(PR12)
PR10                C = 'PL,'                    /(PR12)
PR11                C = TAB ',' SA SB
PR12                PROD = C                     /(RETURN)
PR13    TAB         '*' *X* '*'                  /F(PR3)
                    C = UNION(A,B)
                    EQUALS(A,C)                  /S(PR12)
                    EQUALS(B,C)                  /S(PR12)F(PR3)

UNION

UN1     A           'PL'                         /S(UN12)
        B           'PL'                         /S(UN13)
        A           *TA* ',' *SA*
        B           *TB* ',' *SB*
                    TAB = TA TB
        TAB         'L'                          /S(UN21)
        TAB         '*' *X* '*'                  /S(UN14)
        TA          '*'                          /S(UN15)
        TB          '*'                          /S(UN16)
                    EQUALS(SA,SB)                /S(UN22)
UN2                 X = 'A'
UN3     $('T' X)    '*'                          /F(UN6)
        $('T' X)    'I'                          /S(UN4)
                    $('Y' X) = ',(' $('S' X) ')*,'         /(UN5)
UN4                 $('Y' X) = ',' $('S' X) '*,'
UN5     X           'A' = 'B'                    /S(UN3)F(UN8)
UN6                 $('S' X) = $('S' X) 'U'
                    $('Y' X) = ','
UN7     $('S' X)    *(Z)* 'U' =                  /F(UN5)
                    $('Y' X) = $('Y' X) Z ','              /(UN7)
UN8                 C =
UN9     YA          ',' *Z* ',' = ','            /F(UN10)
        YB          ',' Z ','                    /S(UN9)
                    C = C ',' Z                  /(UN9)
UN10                C = C YB 'E'
        C           ',' *Z* ',F' = Z
UN11    C           ',' = 'U'                    /S(UN11)
        TAB         'I'                          /S(UN23)
        TAB         'E'                          /S(UN24)
                    C = 'S,' C                   /(UN25)
UN12                C = B                        /(UN25)
UN13                C = A                        /(UN25)
UN14                .GE(SIZE(SA),SIZE(SB))       /F(UN16)
UN15                YA = 'A'
                    YB = 'B'                     /(UN17)
UN16                YA = 'B'
                    YB = 'A'
UN17                .GE(SIZE($('S' YA)),SIZE($('S' YB)))   /F(UN2)
                    Z = ','
                    ZA = $('S' YA) 'U'
                    ZB = $('S' YB) 'U'
UN18    ZA          *(X)* 'U' =                  /F(UN19)
                    Z = Z X ','                  /(UN18)
UN19    ZB          *(X)* 'U' =                  /F(UN20)
        Z           ',' X ','                    /S(UN19)F(UN2)
```

UNION (CON'T)

```
UN20                    C = $YA                         /(UN25)
UN21                    C = TAB ',' SA SB
          C             'L' =
          C             'L'                             /S(UN25)
          C             'I'                             /S(UN25)
          C             'E'                             /S(UN25)
          C             *X* ',' *Z* = X 'E,' Z          /(UN25)
UN22      TA            'I'                             /S(UN13)
          TB            'I'                             /S(UN12)
          TA            'E'                             /S(UN13)F(UN12)
UN23                    C = 'SI,' C                     /(UN25)
UN24                    C = 'SE,' C
UN25                    UNION = C                       /(RETURN)
END
```

V. Display the Lattice of SP Partitions

A. Program name: SP

B. Purpose: To display in a convenient manner the lattice of SP partitions for a specified finite-state machine.

C. Method: For a given machine the set of two-state generators, S_2, is formed as follows:

$$S_2 = \bigcup_{i,j \in S} \min \{\pi \mid \pi \text{ is an SP partition on } S \text{ such that } \pi(i) = \pi(j)\};$$

A, the set of lattice atoms, is

$$A = \min S_2;$$

B, the set of basic generators, is extracted from S_2 and A via:

$$B = A \cup \{\pi \mid \pi \in S_2 - A, \pi \neq \sum_{\substack{\pi' \in S, \\ \pi' < \pi}} \pi'\}$$

The lattice is then generated in successive rows, R(0), R(1), ..., etc. ... where

R(0) = {zero partition}

R(1) = A

⋮

$$\vdots$$

$$R(k) = \min \left\{ \pi \;\middle|\; \begin{array}{l} \pi \in B - R'(k-1) \text{ or} \\ \pi = \pi_1 + \pi_2 \text{ for some } \pi_1 \neq \pi_2, \\ \qquad \pi_1, \pi_2 \notin R'(k-1) \text{ such} \\ \qquad \text{that } \pi_1 + \pi_2 \in R'(k-1) \end{array} \right\}$$

$$\text{where } R'(k-1) = \sum_{\ell=0}^{k-1} R(\ell)$$

$$\vdots$$

$$R(m) = \{\text{the identity partition}\}$$

The partitions are outputted row-by-row and are identified by number—zero designating the zero partition and the numbers 1,2,3,... etc. designating successively higher partitions in the lattice with the highest number belonging to the identity partition. As the partitions in each row are identified their identification number, row number, lattice type, immediate successor identification numbers and non-singleton partition blocks are listed. Finally, the two-state partitions induced by all state pairs are tabulated.

D. Language and System: IBM System/360 FORTRAN IV to be run in the time-sharing mode on MTS (Michigan Terminal System) as it existed at the University of Michigan, July 1967.

E. Operation: For on-line time-sharing execution run the object version of the program with 1 = *SOURCE*
2 = *SINK*

All user input is supplied at program request in the following sequence:

Program Types	User Types
1) "MACHINE NAME? (TYPE UP TO 50 CHARACTERS)"	Any string of characters up to length 50 to identify the machine about to be described.
2) "N? (TYPE A 3-DIGIT NUMBER IN RANGE 1 TO 100)"	A single 3-digit integer corresponding to N for the machine being described; $1 \leq N \leq 100$; the number will be read by the format I3. If N is out of range the program will ask for N again.
3) "P? (TYPE A 1-DIGIT NUMBER IN RANGE 1 TO 5)"	A single 1-digit integer corresponding to P for the machine being described; $1 \leq P \leq 5$; the number will be read by the format I1. If P is out of range the program will ask for P again.

4) "STATE TRANSITION TABLE: FOR EACH I TYPE P 3-DIGIT NUMBERS SEPARATED BY COMMAS AND CORRESPONDING TO FS(I, J) FOR J=1 to P

I = 1"	P 3-digit integers separated by commas and corresponding to FS(1, 1), FS(1, 2), ..., FS(1, P); these numbers will be read by the format 5(3I, 1X).
"I = 2"	P 3-digit integers separated by commas and corresponding to FS(2, 1), FS(2, 2), ..., FS(2, P); these numbers will be read by the format 5(3I, 1X).
⋮	⋮
"I = N"	P 3-digit integers corresponding to FS(N, 1), FS(N, 2), ..., FS(N, P).

5) Re-capitulation of NAME, N, P and full TRANSITION TABLE information.

6) LATTICE TABLE in which each lattice point representing a partition with SP on the given machine is listed; each point is given an identification number; the row in the lattice of each point is indicated; the type for each point is specified according as the point is a lattice atom, a basic generator, a two-state generator or none of the above; the identification numbers of all points immediately less than each point are given; the non-singleton blocks of each partition are enumerated (singleton blocks will be omitted from the output).

7) TWO-STATE GENERATOR TABLE in which is tabulated a list of the minimal SP partitions induced by each pair of states in the machine. The partition number refers to the

identification numbers listed
in the LATTICE TABLE above.

8) "NEW MACHINE (0=NO, 1=YES)?"	Zero or one as desired; number will be read by the format I1.

F. Sample Run

```
#$RUN SPOBJ;1=*SOURCE* 2=*SINK*
. IBCOM#   IS AN UNDEFINED SYMBOL.
#EXECUTION BEGINS


 SP LATTICE PROGRAM


 MACHINE NAME?(TYPE UP TO 50 CHARACTERS)
 HARTMANIS AND STEARNS,PAGE 42,MACHINE B

 N?(TYPE A 3-DIGIT NUMBER IN RANGE 1 TO 100)
 008

 P?(TYPE A 1-DIGIT NUMBER IN RANGE 1 TO 5)
 2


 STATE TRANSITION TABLE:
 FOR EACH I TYPE 2 3-DIGIT NUMBERS
 SEPARATED BY COMMAS AND
 CORRESPONDING TO FS(I,J)
 FOR J=1 TO  2

 I =   1
 003,007

 I =   2
 004,008

 I =   3
 001,006

 I =   4
 002,005

 I =   5
 002,004

 I =   6
 001,003

 I =   7
 004,004

 I =   8
 003,003


  --------------------------------------


 SP LATTICE PROGRAM

 MACHINE NAME = HARTMANIS AND STEARNS,PAGE 42,MACHINE B

 N =   8     P =   2


  --------------------------------------
```

```
STATE TRANSITION TABLE

            INPUTS
STATE        1    2

  1          3    7
  2          4    8
  3          1    6
  4          2    5
  5          2    4
  6          1    3
  7          4    4
  8          3    3


 ----------------------------------------

LATTICE TABLE

TYPE CODE: A=LATTICE ATOM
           B=BASIC GENERATOR
           2=TWO-STATE GENERATOR

NO.  ROW  TYPE

 0    0  ZERO

 1    1  AB2   SUCC:   0

                 BLOCK   1:   1   2

                 BLOCK   2:   3   4

                 BLOCK   3:   5   6

                 BLOCK   4:   7   8

 2    1  AB2   SUCC:   0

                 BLOCK   3:   3   6

 3    1  AB2   SUCC:   0

                 BLOCK   4:   4   5

 4    2   B2   SUCC:   1

                 BLOCK   1:   1   2   3   4

                 BLOCK   2:   5   6   7   8

 5    2        SUCC:   2   3

                 BLOCK   3:   3   6

                 BLOCK   4:   4   5

 6    3    2   SUCC:   1   5

                 BLOCK   1:   1   2

                 BLOCK   2:   3   4   5   6

                 BLOCK   3:   7   8

 7    4    2   SUCC:   4   6

                 BLOCK   1:   1   2   3   4   5   6   7   8


 ----------------------------------------
```

```
TWO-STATE GENERATOR TABLE

     STATE STATE PARTITION NO.

       1      2      1
       1      3      4
       1      4      4
       1      5      7
       1      6      7
       1      7      7
       1      8      7
       2      3      4
       2      4      4
       2      5      7
       2      6      7
       2      7      7
       2      8      7
       3      4      1
       3      5      6
       3      6      2
       3      7      7
       3      8      7
       4      5      3
       4      6      6
       4      7      7
       4      8      7
       5      6      1
       5      7      4
       5      8      4
       6      7      4
       6      8      4
       7      8      1


 --------------------------------------

NEW MACHINE(0=NO, 1=YES)?
1


MACHINE NAME?(TYPE UP TO 50 CHARACTERS)
FARR,JOUR OF ACM,JULY 1963,PAGE 382,MACHINE B

N?(TYPE A 3-DIGIT NUMBER IN RANGE 1 TO 100)
745
***N OUT OF RANGE

N?(TYPE A 3-DIGIT NUMBER IN RANGE 1 TO 100)
010

P?(TYPE A 1-DIGIT NUMBER IN RANGE 1 TO 5)
7
***P OUT OF RANGE

P?(TYPE A 1-DIGIT NUMBER IN RANGE 1 TO 5)
2
```

```
STATE TRANSITION TABLE:
FOR EACH I TYPE 2 3-DIGIT NUMBERS
SEPARATED BY COMMAS AND
CORRESPONDING TO FS(I,J)
FOR J=1 TO  2

I =  1
001,006

I =  2
001,007

I =  3
001,007

I =  4
001,008

I =  5
002,008

I =  6
003,009

I =  7
003,010

I =  8
004,010

I =  9
004,010

I =  10
005,010
```

```
SP LATTICE PROGRAM

MACHINE NAME = FARR,JOUR OF ACM,JULY 1963,PAGE 382,MACHINE B

N =  10     P =  2
```

STATE TRANSITION TABLE

STATE	INPUTS 1	2
1	1	6
2	1	7
3	1	7
4	1	8
5	2	8
6	3	9
7	3	10
8	4	10
9	4	10
10	5	10

```
LATTICE TABLE

TYPE CODE: A=LATTICE ATOM
           B=BASIC GENERATOR
           2=TWO-STATE GENERATOR

NO.  ROW  TYPE

 0    0   ZERO

 1    1   AB2   SUCC:  0
                   BLOCK  1:  1  2
                   BLOCK  3:  4  5
                   BLOCK  4:  6  7
                   BLOCK  6:  9 10

 2    1   AB2   SUCC:  0
                   BLOCK  2:  2  3

 3    1   AB2   SUCC:  0
                   BLOCK  3:  3  4
                   BLOCK  6:  7  8

 4    1   AB2   SUCC:  0
                   BLOCK  8:  8  9

 5    2    2    SUCC:  1  2
                   BLOCK  1:  1  2  3
                   BLOCK  2:  4  5
                   BLOCK  3:  6  7
                   BLOCK  5:  9 10

 6    2    2    SUCC:  2  3
                   BLOCK  2:  2  3  4
                   BLOCK  5:  7  8

 7    2    2    SUCC:  1  3
                   BLOCK  1:  1  2
                   BLOCK  2:  3  4  5
                   BLOCK  3:  6  7  8
                   BLOCK  4:  9 10

 8    2    2    SUCC:  3  4
                   BLOCK  3:  3  4
                   BLOCK  6:  7  8  9

 9    2    2    SUCC:  1  4
                   BLOCK  1:  1  2
                   BLOCK  3:  4  5
                   BLOCK  4:  6  7
                   BLOCK  5:  8  9 10
```

```
10    2         SUCC:   2   4
                   BLOCK    2:   2   3
                   BLOCK    7:   8   9
11    3    2    SUCC:   5   6   7
                   BLOCK    1:   1   2   3   4   5
                   BLOCK    2:   6   7   8
                   BLOCK    3:   9  10
12    3   B2    SUCC:  10
                   BLOCK    2:   2   3
                   BLOCK    4:   5   6
                   BLOCK    6:   8   9
13    3    2    SUCC:   7   8   9
                   BLOCK    1:   1   2
                   BLOCK    2:   3   4   5
                   BLOCK    3:   6   7   8   9  10
14    3         SUCC:   5   9  10
                   BLOCK    1:   1   2   3
                   BLOCK    2:   4   5
                   BLOCK    3:   6   7
                   BLOCK    4:   8   9  10
15    3         SUCC:   6   8  10
                   BLOCK    2:   2   3   4
                   BLOCK    5:   7   8   9
16    4    2    SUCC:  12  14
                   BLOCK    1:   1   2   3
                   BLOCK    2:   4   5   6   7
                   BLOCK    3:   8   9  10
17    4         SUCC:  11  13  14  15
                   BLOCK    1:   1   2   3   4   5
                   BLOCK    2:   6   7   8   9  10
18    4         SUCC:  12  15
                   BLOCK    2:   2   3   4
                   BLOCK    3:   5   6
                   BLOCK    4:   7   8   9
19    5    2    SUCC:  16  17  18
                   BLOCK    1:   1   2   3   4   5   6   7   8   9  10

-------------------------------------
```

TWO-STATE GENERATOR TABLE

STATE	STATE	PARTITION NO.
1	2	1
1	3	5
1	4	11
1	5	11
1	6	19
1	7	19
1	8	19
1	9	19
1	10	19
2	3	2
2	4	6
2	5	11
2	6	19
2	7	19
2	8	19
2	9	19
2	10	19
3	4	3
3	5	7
3	6	19
3	7	19
3	8	19
3	9	19
3	10	19
4	5	1
4	6	16
4	7	16
4	8	19
4	9	19
4	10	19
5	6	12
5	7	16
5	8	19
5	9	19
5	10	19
6	7	1
6	8	7
6	9	13
6	10	13
7	8	3
7	9	8
7	10	13
8	9	4
8	10	9
9	10	1

NEW MACHINE(0=NO, 1=YES)?
0
IHC002I STOP 0 ***** RESTART AT LOCATION 0627D2
EXECUTION TERMINATED

G. Important Program Variables:

EQUAL(N, PPM, TP1, PP, LEQ, PPEQ) is a subroutine which scans the partitions in the PP array comparing them with the partition in TP1; if a match is found then a return is made with LEQ = 1 and PPEQ = the address of the PP partition identical to the TP1 partition; if no match is found then a return is made with LEQ = 0; all partitions must be normalized and sized; rank, number and type codes are ignored.

LEQ (See EQUAL)

LESS (J, I, N, PP) is a logical function whose values is .TRUE. iff the partition with address J in PP is less than or equal to the partition with address I.

NORSIZ (N, TP1) is a subroutine which normalizes and sizes the partition given in TP1.

PP is a linear array in which partition information is stored; each partition occupies a segment of length N+4 coded as follows:

Segment Cell	Code
1	rank: $-1 \rightarrow$ old partition $0 \rightarrow$ present partition $\geq 1 \rightarrow$ future partition
2	number: $< 0 \rightarrow$ temporary ID $= 0 \rightarrow$ zero partition $> 0 \rightarrow$ final ID
3	type: $1 \rightarrow$ basic generator $2 \rightarrow$ two-state generator $3 \rightarrow$ none of the above
4	size: the number of blocks in the indicated partition
5, 6, . . . , N+4	correspond to the N states of the machine; two cells contain the same number iff their corresponding states are in the same block of the partition being described; when in normal form cell 5, the cell corresponding to state 1, will contain the number '1'; the cell corresponding to the least state not in the same block as state 1 will contain '2', etc. . . . . The address of the segment corresponds to the N+4 cell.

PPEQ (See EQUAL)

PPM = the index of the last cell of the last partition in the PP array

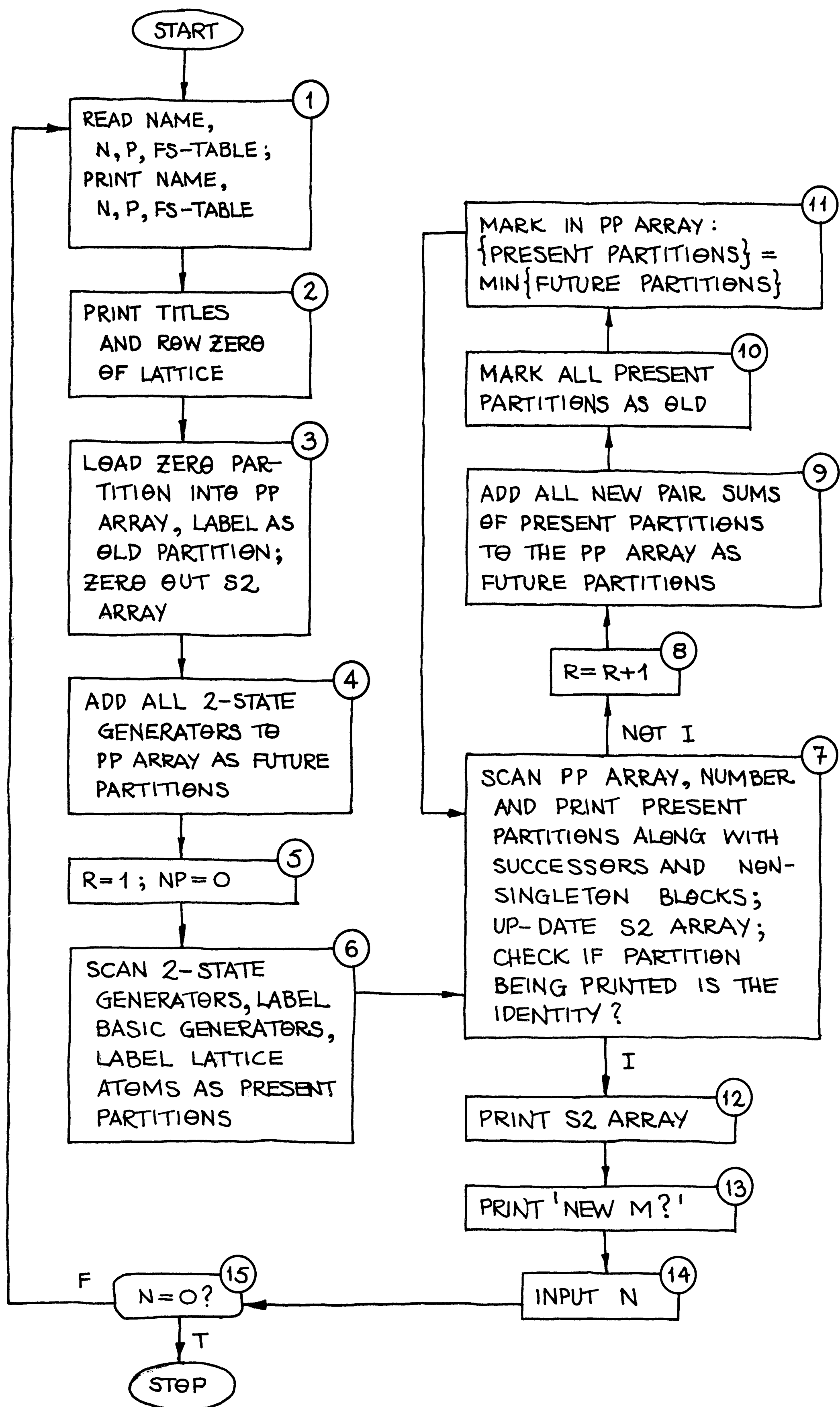
REDUCE(N, P, FS, TP1) is a subroutine which replaces the partition in TP1 with the smallest partition with SP that contains it; the partition in TP1 need not be in normal form nor need it be ranked, numbered, typed or sized.

S2 is a two-dimensional array; S2(I, J) is the number (either temporary or final) of the two-state generator partition obtained by equivalencing states I and J; if S2(I, J) = 0 then this partition is as yet unknown.

SUM (N, TP1, TP2) is a subroutine which places the sum of the partitions in TP1 and TP2 into TP1; rank, number, type and size are ignored; the initial and final partitions need not be in normal form.

TP1 and TP2 are two linear arrays in each of which temporary information on a single partition may be stored; the format is the same as a single PP array segment.

H. Annotated Flow Chart



I. Annotated Program Listing

```
#$COPY SP TO *SINK*
       IMPLICIT INTEGER*2(A-Z)
       REAL*8 TYPE
       LOGICAL LESS
       DIMENSION FS(100,5),NAME(50),PP(5000),S2(100,100)
       DIMENSION SUCC(100),TP1(104),TP2(104),TYPE(4)
       DATA Q/'Q'/,TYPE/'AB2',' B2',' 2',' '/   <- FIRST VARIABLE
       WRITE(2,10)                                 Q IS A DUMMY
    10 FORMAT(//18HSP LATTICE PROGRAM)
    20 WRITE(2,30)
    30 FORMAT(//39HMACHINE NAME?(TYPE UP TO 50 CHARACTERS))
       READ(1,40)NAME
    40 FORMAT(50A1)
    50 WRITE(2,60)
    60 FORMAT(/43HN?(TYPE A 3-DIGIT NUMBER IN RANGE 1 TO 100))
       READ(1,70)N
    70 FORMAT(I3)
       IF(N*(101-N))80,80,100
    80 WRITE(2,90)
    90 FORMAT(17H***N OUT OF RANGE)
       GO TO 50
   100 WRITE(2,110)
   110 FORMAT(/41HP?(TYPE A 1-DIGIT NUMBER IN RANGE 1 TO 5))
       READ(1,120)P
   120 FORMAT(I1)
       IF(P*(6-P))130,130,150
   130 WRITE(2,140)
   140 FORMAT(17H***P OUT OF RANGE)
       GO TO 100
   150 WRITE(2,160)
   160 FORMAT(//23HSTATE TRANSITION TABLE:)
       WRITE(2,170)P
   170 FORMAT(15HFOR EACH I TYPE,I2,16H 3-DIGIT NUMBERS)
       WRITE(2,171)
 1 171 FORMAT(23HSEPARATED BY COMMAS AND)
       WRITE(2,172)
   172 FORMAT(24HCORRESPONDING TO FS(I,J))
       WRITE(2,180)P
   180 FORMAT(10HFOR J=1 TO,I3)
       DO 200 I=1,N
       WRITE(2,190)I
   190 FORMAT(/4HI = ,I3)
   200 READ(1,210)(FS(I,J),J=1,P)
   210 FORMAT(5(I3,1X))
       WRITE(2,213)
       WRITE(2,10)
       WRITE(2,211)NAME
   211 FORMAT(/15HMACHINE NAME = ,50A1)
       WRITE(2,212)N,P
   212 FORMAT(/4HN = ,I3,5X,4HP = ,I3)
       WRITE(2,213)
   213 FORMAT(/40(1H-))
       WRITE(2,214)
   214 FORMAT(/22HSTATE TRANSITION TABLE)
       WRITE(2,220)(I,I=1,P)
   220 FORMAT(/11X,6HINPUTS/5HSTATE,3X,5I5)
       WRITE(2,221)
   221 FORMAT(1H )
       DO 230 I=1,N
   230 WRITE(2,240)I,(FS(I,J),J=1,P)
   240 FORMAT(I4,4X,5I5)
       WRITE(2,213)
       WRITE(2,250)
   250 FORMAT(/13HLATTICE TABLE)
       WRITE(2,251)
   251 FORMAT(/25HTYPE CODE: A=LATTICE ATOM)
       WRITE(2,252)
 2 252 FORMAT(11X,17HB=BASIC GENERATOR)
       WRITE(2,253)
   253 FORMAT(11X,21H2=TWO-STATE GENERATOR)
       WRITE(2,260)
   260 FORMAT(/14HNO.  ROW  TYPE)
       WRITE(2,270)
   270 FORMAT(/14H  0    0  ZERO)
```

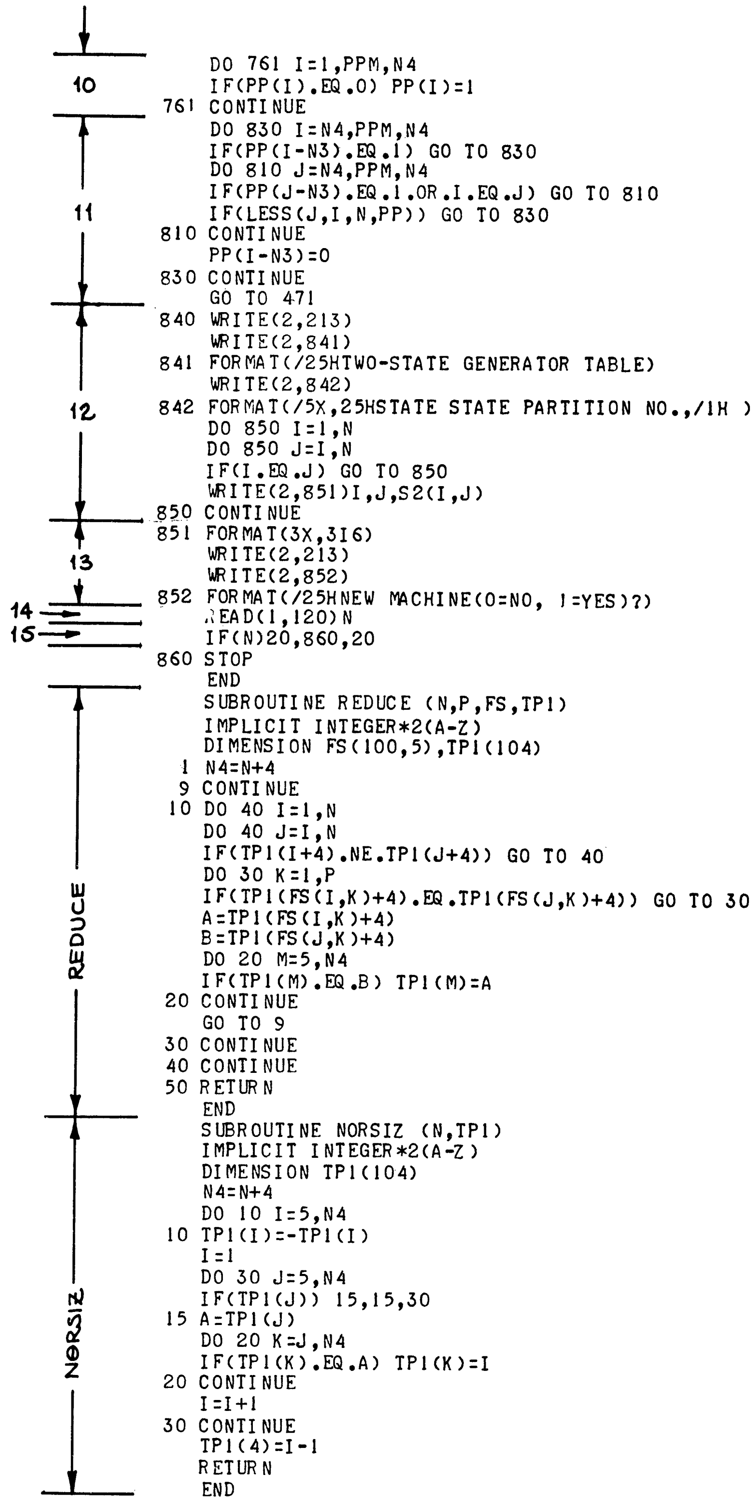
```
  271 PP(1)=1
      PP(2)=0
      PP(3)=3
      PP(4)=N
      N3=N+3
 3    N4=N+4
      PPM=N4
      PN=-1
      DO 280 I=1,N
      PP(I+4)=I
      DO 280 J=I,N
  280 S2(I,J)=0
      DO 400 I=1,N
      DO 400 J=I,N
      IF(I.EQ.J) GO TO 400
      DO 290 K=1,N
  290 TP1(K+4)=K
      TP1(J+4)=I
      DO 370 K=1,P
      IF(FS(I,K)-FS(J,K)) 298,300,297
  297 S2T=S2(FS(J,K),FS(I,K))
      GO TO 299
  298 S2T=S2(FS(I,K),FS(J,K))
  299 IF(S2T) 320,300,320
  300 DO 310 M=1,N
  310 TP2(M+4)=M
      TP2(FS(J,K)+4)=FS(I,K)
      GO TO 360
  320 DO 340 M=2,PPM,N4
      IF(PP(M)-S2T)340,330,340
  330 MT=M-2
      GO TO 350
 4340 CONTINUE
  350 DO 355 M=5,N4
  355 TP2(M)=PP(MT+M)
  360 CALL SUM(N,TP1,TP2)
  370 CONTINUE
      CALL REDUCE(N,P,FS,TP1)
      CALL NORSIZ(N,TP1)
      CALL EQUAL(N,PPM,TP1,PP,LEQ,PPEQ)
      IF(LEQ)390,390,380
  380 S2(I,J)=PP(PPEQ-N-2)
      GO TO 400
  390 DO 395 K=4,N4
  395 PP(PPM+K)=TP1(K)
      PP(PPM+3)=2
      PP(PPM+2)=PN
      PP(PPM+1)=0
      S2(I,J)=PN
      PN=PN-1
      PPM=PPM+N4
  400 CONTINUE
 5    R=1
      NP=0
      N2=2*N+8
      DO 470 I=N2,PPM,N4
      DO 405 J=1,N
  405 TP1(J+4)=J
      S=0
      DO 430 J=N2,PPM,N4
      IF(J.EQ.I.OR.PP(I-N).GE.PP(J-N)) GO TO 430
      IF(.NOT.(LESS(J,I,N,PP))) GO TO 430
      S=1
 6    JT=J-N4
      DO 420 K=5,N4
  420 TP2(K)=PP(JT+K)
      CALL SUM(N,TP1,TP2)
  430 CONTINUE
      IF(S)440,450,440
  440 CALL NORSIZ(N,TP1)
      PP(I-N-3)=-1
      IF(TP1(4)-PP(I-N))450,470,450
  450 PP(I-N-1)=1
  470 CONTINUE
```

7

3

9

```
471 DO 641 I=1,PPM,N4
    IF(PP(I))641,480,641
480 NP=NP+1
    S=PP(I+1)
    DO 500 J=1,N
    DO 500 K=J,N
    IF(S2(J,K)-S)500,490,500
490 S2(J,K)=NP
500 CONTINUE
    PP(I+1)=NP
    IT=I+N3
    S=0
    DO 510 J=1,PPM,N4
    IF(PP(J).NE.1) GO TO 510
    JT=J+N3
    IF(.NOT.LESS(JT,IT,N,PP)) GO TO 510
    PP(J)=2
510 CONTINUE
    DO 530 J=1,PPM,N4
    IF(PP(J).NE.2) GO TO 530
    JT=J+N3
    DO 520 K=1,PPM,N4
    IF(PP(K).NE.2.OR.K.EQ.J) GO TO 520
    KT=K+N3
    IF(LESS(JT,KT,N,PP)) GO TO 525
520 CONTINUE
    S=S+1
    SUCC(S)=PP(J+1)
    GO TO 530
525 PP(J)=1
530 CONTINUE
    DO 535 J=1,PPM,N4
    IF(PP(J).EQ.2) PP(J)=1
535 CONTINUE
    IF(R-1)550,540,550
540 T=1
    GO TO 600
550 IF(PP(I+2)-1)570,560,570
560 T=2
    GO TO 600
570 IF(PP(I+2)-2)590,580,590
580 T=3
    GO TO 600
590 T=4
600 WRITE(2,601)NP,R,TYPE(T),(SUCC(J),J=1,S)
601 FORMAT(/I3,I5,2X,A3,3X,5HSUCC:,10I4,(/21X,10I4))
610 JP=PP(I+3)
    DO 640 J=1,JP
    S=0
    DO 630 K=1,N
    IF(PP(I+3+K)-J)630,620,630
620 S=S+1
    SUCC(S)=K
630 CONTINUE
    IF(S-1)640,640,635
635 WRITE(2,636)J,(SUCC(K),K=1,S)
636 FORMAT(/18X,6HBLOCK ,I3,1H:,10I4,(/28X,10I4))
640 CONTINUE
    IF(PP(I+3).EQ.1) GO TO 840
641 CONTINUE
642 R=R+1
    DO 760 I=1,PPM,N4
    IF(PP(I))760,700,760
700 IT=I+N3
    DO 759 J=I,PPM,N4
    IF(I.EQ.J.OR.PP(J).NE.0) GO TO 759
    DO 720 K=4,N3
    TP1(K+1)=PP(I+K)
720 TP2(K+1)=PP(J+K)
    CALL SUM(N,TP1,TP2)
    CALL NORSIZ(N,TP1)
    CALL EQUAL(N,PPM,TP1,PP,LEQ,PPEQ)
    IF(LEQ)759,740,759
740 DO 750 K=4,N4
750 PP(PPM+K)=TP1(K)
    PP(PPM+1)=-1
    PP(PPM+2)=PN
    PN=PN-1
    PP(PPM+3)=3
    PPM=PPM+N4
759 CONTINUE
760 CONTINUE
```




```
      DO 761 I=1,PPM,N4
      IF(PP(I).EQ.0) PP(I)=1
  761 CONTINUE
      DO 830 I=N4,PPM,N4
      IF(PP(I-N3).EQ.1) GO TO 830
      DO 810 J=N4,PPM,N4
      IF(PP(J-N3).EQ.1.OR.I.EQ.J) GO TO 810
      IF(LESS(J,I,N,PP)) GO TO 830
  810 CONTINUE
      PP(I-N3)=0
  830 CONTINUE
      GO TO 471
  840 WRITE(2,213)
      WRITE(2,841)
  841 FORMAT(/25HTWO-STATE GENERATOR TABLE)
      WRITE(2,842)
  842 FORMAT(/5X,25HSTATE STATE PARTITION NO.,/1H )
      DO 850 I=1,N
      DO 850 J=I,N
      IF(I.EQ.J) GO TO 850
      WRITE(2,851)I,J,S2(I,J)
  850 CONTINUE
  851 FORMAT(3X,3I6)
      WRITE(2,213)
      WRITE(2,852)
  852 FORMAT(/25HNEW MACHINE(0=NO, 1=YES)?)
      READ(1,120)N
      IF(N)20,860,20
  860 STOP
      END
      SUBROUTINE REDUCE (N,P,FS,TP1)
      IMPLICIT INTEGER*2(A-Z)
      DIMENSION FS(100,5),TP1(104)
    1 N4=N+4
    9 CONTINUE
   10 DO 40 I=1,N
      DO 40 J=I,N
      IF(TP1(I+4).NE.TP1(J+4)) GO TO 40
      DO 30 K=1,P
      IF(TP1(FS(I,K)+4).EQ.TP1(FS(J,K)+4)) GO TO 30
      A=TP1(FS(I,K)+4)
      B=TP1(FS(J,K)+4)
      DO 20 M=5,N4
      IF(TP1(M).EQ.B) TP1(M)=A
   20 CONTINUE
      GO TO 9
   30 CONTINUE
   40 CONTINUE
   50 RETURN
      END
      SUBROUTINE NORSIZ (N,TP1)
      IMPLICIT INTEGER*2(A-Z)
      DIMENSION TP1(104)
      N4=N+4
      DO 10 I=5,N4
   10 TP1(I)=-TP1(I)
      I=1
      DO 30 J=5,N4
      IF(TP1(J)) 15,15,30
   15 A=TP1(J)
      DO 20 K=J,N4
      IF(TP1(K).EQ.A) TP1(K)=I
   20 CONTINUE
      I=I+1
   30 CONTINUE
      TP1(4)=I-1
      RETURN
      END
```

EQUAL

```
      SUBROUTINE EQUAL (N,PPM,TP1,PP,LEQ,PPEQ)
      IMPLICIT INTEGER*2(A-Z)
      DIMENSION TP1(104),PP(5000)
      N4=N+4
      DO 20 I=N4,PPM,N4
      DO 10 J=4,N4
      IF(TP1(J).NE.PP(I-N-4+J)) GO TO 20
      CONTINUE
      LEQ=1
      PPEQ=I
      GO TO 30
   20 CONTINUE
      LEQ=0
   30 RETURN
      END
```

SUM

```
      SUBROUTINE SUM (N,TP1,TP2)
      IMPLICIT INTEGER*2(A-Z)
      DIMENSION TP1(104),TP2(104)
      N4=N+4
      DO 40 I=5,N4
      IF(TP2(I).EQ.0) GO TO 40
      A=TP2(I)
      DO 30 J=I,N4
      IF(TP2(J).NE.A) GO TO 30
      IF(TP1(I).EQ.TP1(J)) GO TO 20
      B=TP1(I)
      C=TP1(J)
      DO 10 K=5,N4
      IF(TP1(K).EQ.C) TP1(K)=B
   10 CONTINUE
   20 TP2(J)=0
   30 CONTINUE
   40 CONTINUE
      RETURN
      END
```

LESS

```
      LOGICAL FUNCTION LESS(J,I,N,PP)
      IMPLICIT INTEGER*2(A-Z)
      DIMENSION PP(5000)
      IT=I-N
      JT=J-N
      DO 10 K=1,N
      DO 10 M=K,N
      IF(PP(JT+K).EQ.PP(JT+M).AND.PP(IT+K).NE.PP(IT+M)) GO TO 20
   10 CONTINUE
      LESS=.TRUE.
      RETURN
   20 LESS=.FALSE.
      RETURN
      END
```

VI. Bibliography

General reference on finite-state machines and in particular on state diagnosing:

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

Regular Expresssions :

[2] McNaughton, R. and Yamada, H., Regular Expressions and State Graphs for Automata, *IRE Transactions on Electronic Computers*, vol. EC-9, no. 1 (March 1960), pp. 39-47.

[3] Harrison, M. A., *Introduction to Switching and Automata Theory*, McGraw-Hill, New York (1965).

[4] Brzozowski, J. A., and McCluskey, Signal Flow Graph Techniques for Sequential Circuit State Diagrams, *IEEE Transactions on Electronic Computers*, vol. EC-12, no. 2 (April 1963), pp. 67-76.

SP Lattices:

[5] Hartmanis, J. and Stearns, R. E., *Algebraic Structure Theory of Sequential Machines*, Prentice Hall, New Jersey (1966).

References to computer programs dealing with finite state machines:

[6] Farr, E. H., Lattice Properties of Sequential Machines, *Journal of the Association for Computing Machinery*, July 1963, vol. 10, no. 3, p. 365.

This paper mentions a program to produce all the partitions with S. P. for a given machine for which $N \leq 38$, $P \leq 40$.

[7] Griffiths, T. V., M-460 Program Notes: Some LISP Routines for Manipulating Automata, AFCRL-66-84, February 1967, Physical and Mathematical Sciences Research Papers, no. 192, Data Sciences Laboratory Project 4641, Air Force Cambridge Research Laboratories, L. G. Hanscom Field, Bedford, Massachusetts.

This paper details a number of LISP programs to represent machines, cascade them, minimize them, complement them, compute their behavioral union, intersection, star, and reverse, etc.

[8] Roberts, M. B., A Generalized Recognizer for Finite State Languages, Moore School of Electrical Engineering, University of Pennsylvania Report no. 66-03, August 1965.

This report discusses an IPL-V program to determine if a given symbol string is denoted by a given regular expression; programs to convert a regular expression to a state table and minimize it are also mentioned.

[9] Silverstein, M., Computer Procedures for Analysis of Finite Automata, term project, University of California, Berkeley (1962).

This paper mentioned IPL-V routines to minimize a given machine, develop distinguishing sequences for state pairs, determine multiple preset diagnosing and regular preset homing experiments (neither of these necessarily minimal).

[10] Sarma, Hota S., *Design of a Computer Program for Minimization and State Identification of Automata*, Masters thesis, Electrical Engineering, Tuskegee Institute, May 1966.

This thesis describes a program written for online use of an IBM 1620 and which minimizes a given machine and determines minimal simple preset homing and diagnosing experiments for a given admissible set.

[11] Sacco, W. J., A Computer Technique Useful for Some Problems in the Partitioning Theory of Sequential Machines, Memorandum Report no. 1733, March 1966, Ballistic Research Laboratories, U. S. Army Material Command, Aberdeen Proving Ground, Maryland.

This paper gives no computer program but rather a technique for determining if (1) a given partition has SP, (2) two given partitions constitute a partition pair, and (3) two set systems constitute a system pair.

addition, W. D. Maurer uses a computer to assist in minimally decomposing group machines; a report on his work is to appear in a new journal, *The Journal of Computer and Systems Science*.

UNIVERSITY OF MICHIGAN
3 9015 03695 2060