# LOGIC-FREE REALIZATIONS OF SEQUENTIAL MACHINES

by

Peter Hawtrey, B.A.Sc.

Submitted in partial fulfillment of the requirements
for the degree of Master of Science.



Department of Electrical Engineering
Faculty of Pure and Applied Science
The University of Ottawa
Ottawa, CANADA
July, 1966.

To:

John Ciardi, who saw the ogre's eye;
Morgan, who laughed, and Kate who
laughed too, but didn't know why.

L's and J's, G's and P's and the others
who passed as shadows.

And finally, Instinct, who, in spite
of them, saw me through.

# LOGIC-FREE REALIZATIONS OF SEQUENTIAL MACHINES

## ABSTRACT

Ultimately, the abstract sequential machine must be implemented with hardware, and the designer must decide what circuit structure best suits his needs. He is limited in his choice, however, for there exists a definite relationship between a specific circuit structure and the type of machine behaviour it will realize; this relationship must be known if the designer is to chose a realistic structure for the circuit which must realize his machine. It is the purpose of this paper to characterize those machines which can be realized by circuit structures that are logic-free - that is, which consists solely of memory elements.

In general, a circuit with no logic can be divided into the parallel combination of up to three component circuits; a constant input circuit with no feedback, a constant input circuit with feedback, and an input dependent feedback-free circuit. These, in turn, define certain types of machines. By studying the homomorphic images of submachines of such circuits individually, and then collectively, we are able to characterize the behaviour of all machines which can be logic-free realized, and present assignment methods for the realization.

## ACKNOWLEDGMENTS

It was Professor J. A. Brzozowski who suggested the problem. For this and his invaluable help through numerous discussions, comments and criticisms, for his sincere encouragement and proffered advice, I am grateful.

I thank my colleagues S. Singh, J. McEntyre and W. Davis for their advice and encouragement during our seminars, and scintillating conversation after.

I would also like to thank Odette Lapierre for her patience and understanding while typing and correcting the manuscript.

Finally, I am grateful for the financial assistance of Northern Electric R and D Labs, Ottawa, Canada.

TABLE OF CONTENTS

# 1. FUNDAMENTALS

## 1.1 Introduction

A sequential machine is said to be logic-free realizable if and only if there exists a sequential circuit which realizes the machine yet uses only memory elements. Such a realization would be very simple in terms of hardware and possibly minimal in terms of cost. As part of the structure theory of sequential machines, then, this problem should definitely be considered.

In general, a circuit with no logic can be divided into the parallel combination of up to three component circuits; a constant input circuit with no feedback, a constant input circuit with feedback, and an input dependent circuit with no feedback. These, in turn, define certain types of machines. By studying the homomorphic images of submachines of such circuits individually, and then collectively, we are able to characterize the behaviour of all machines which can be logic-free realized, and present assignment methods for the realization.

This paper is divided into 4 sections; the first summarizes the basic concepts used in later developments; the second section studies the logic-free properties of those machines defined by the component circuits. In the third section, parallel state realizations are developed, followed by a study of homomorphic image machines of cyclic submachines. Finally, in the last section, the remaining class of logic-free realizable machines is characterized.

## 1.2. Notation and Basic Definitions

The abstract algebraic model of a sequential machine has led to the application of many principles of modern algebra to the analysis and synthesis of these machines. In this section we will review some of the properties of algebraic structure-preserving maps as applied to sequential machines. For further details see [1] and [2].

Definition 1.1: A sequential machine is a quintuple $M = (S, I, O, \delta, \lambda)$ where S, I, O are respectively the nonvoid finite sets of states, inputs and outputs:

$$\delta: S \times I \longrightarrow S$$

is the next state function, and

$$\lambda: S \times I \longrightarrow O$$

is the output function. [1]

Definition 1.2: A state machine is a triplet $M = (S, I, \delta)$ where S, I, are respectively the nonvoid finite sets of states and inputs:

$$\delta: S \times I \longrightarrow S$$

is the next state function. [1]

Definition 1.3: A submachine $M_\sigma$ of some machine $M = (S, I, O, \delta, \lambda)$ is a quintuple $(S_\sigma, I_\sigma, O_\sigma, \delta_\sigma, \lambda_\sigma)$ such that

$$S_\sigma \subseteq S, \; I_\sigma \subseteq I, \text{ and } O_\sigma \subseteq O$$

(where $A \subseteq B$ means that A is a subset of B), $\delta_\sigma$ and $\lambda_\sigma$ are the next state and output functions restricted to $S_\sigma$, $I_\sigma$ and $O_\sigma$, and $S_\sigma$ and $O_\sigma$ are closed under $\delta_\sigma$ and $\lambda_\sigma$.

In general, the machines will be represented by flow tables whose columns are inputs, whose rows are the present states and whose entries are next states and outputs. We deal only with completely specified machines.

Definition 1.4: Let $I = \{x_1, x_2, \ldots, x_u\}$ denote the input alphabet and $O = \{z_1, z_2, \ldots, z_r\}$ denote the output alphabet. The $x_i$ $(i = 1, \ldots, u)$ are called the letters of the alphabet. The input dictionary, denoted by $I^*$ is defined to be the set of all words formed from I, where a word or tape means any finite sequences of letters. $I^*$ contains the word with no letters, the null word $\Lambda$. $I.I^*$ denotes the set of all words of length greater than or equal to 1.

The length of any word $\bar{x}$ denoted by $\ell(\bar{x})$, is defined as the number of letters in the word, and $\ell(\Lambda) = 0$.

The next state function $\delta$ and output function $\lambda$ are extended to words of length other than one by requiring

1) for all $s \in S$; $\bar{\delta}(s, \Lambda) = s$,

2) for all $s \in S$, $\bar{x} \in I^*$, $x_j \in I$; $\bar{\delta}(s, \bar{x}.x_j) = \delta(\bar{\delta}(s, \bar{x}), x_j)$,

3) for all $s \in S$, $\bar{x} \in I^*$, $x_j \in I$; $\bar{\lambda}(s, \bar{x}.x_j) = \lambda(\bar{\delta}(s, \bar{x}), x_j)$.

## 1.3. Homomorphisms and Congruence Relations

In general a mapping h from an algebraic system A onto another system B is said to be a homomorphism of A onto B if h preserves the algebraic structure of A. By applying this concept to sequential machines we arrive at the following definitions:

Definition 1.5: A machine $M' = (S', I', O', \delta', \lambda')$ is the homomorphic image of a submachine $M_\sigma = (S_\sigma, I_\sigma, O_\sigma, \delta_\sigma, \lambda_\sigma)$ of machine M if and only if there exists three onto homomorphic mappings $h_S, h_I, h_O$ such that

$$h_S: S_\sigma \longrightarrow S', \quad h_I: I_\sigma \longrightarrow I', \quad h_O: O_\sigma \longrightarrow O', \text{ and}$$

$$h_S(\delta_\sigma(s, x)) = \delta'(h_S(s), h_I(x)),$$

$$h_O(\lambda_\sigma(s, x)) = \lambda'(h_S(s), h_I(x)), \text{ for all } s \in S_\sigma, x \in I_\sigma. \quad [1].$$

Conversely, we say that M is a projection machine of M' denoted by $M = P(M')$ under the same conditions. Note that if M and M' are state machines, $h_O$ is simply not defined.

Definition 1.6: A partition R(S) of a set S is a collection of disjoint subsets of S called blocks such that their set union is S.

An equivalence relation R(S) defined on S determines a partition; the two terms will be used synonymously.

The union of two sets will be denoted by $A \cup B$, their intersection $A \cap B$, and the complement of a set A with respect to the universal set, by $\bar{A}$.

The least upper bound, or sum [1] of two partitions $R_1(S)$ and $R_2(S)$ on S will be denoted by $R_1(S) + R_2(S)$, and the greatest lower bound or product by $R_1(S) \cdot R_2(S)$. Thus, if

$$R_1(S) = \overline{1,2;3,4;5,6} \text{ and } R_2(S) = \overline{1,6;2,3;4,5}$$

then

$$R_1(S) + R_2(S) = \overline{1,2,3,4,5,6}$$

$$R_1(S) \cdot R_2(S) = \overline{1;2;3;4;5;6}.$$

For $R_1(S)$ and $R_2(S)$ on S we say that $R_2(S)$ is larger than or equal to $R_1(S)$ and write $R_1(S) \subseteq R_2(S)$ if and only if every block of $R_1(S)$ is contained in a block of $R_2(S)$. [1]

We say that a variable is assigned according to a partition if this variable has constant value over all elements in the same block of the partition. [1]

Definition 1.7: Let $M = (S, I, O, \delta, \lambda)$ be any sequential machine. R(S) is said to be a right congruence relation on S if and only if R(S) is an equivalence relation on S which satisfies the substitution property (S.P.); for all $s_1, s_2 \in S$, $s_1 R(S) s_2 \rightarrow \delta(s_1, x_i)\ R(S)\ \delta(s_2, x_i)$ for all $x_i \in I$. [1]

Lemma 1.1: Every homomorphism $h = (h_S, h_I, h_O)$ of $M = (S, I, O, \delta, \lambda)$ onto $M' = (S', I', O', \delta', \lambda')$ induces a congruence relation or S.P. partition R(S) on S, and equivalence relations R(I) and R(O) on I and O respectively. Conversely, every congruence relation R(S) on S defines a homomorphism $h = (h_S: S \rightarrow S',\ h_I: I \rightarrow I',\ h_O: O \rightarrow O')$.

Proof: Given $h_S, h_I, h_O$ define $w_i R(W) w_j \leftrightarrow h_W(w_i) = h_W(w_j)$ for $W \in \{S, I, O\}$ and $w_i, w_j \in \{S, I, O\}$. It is trivial to verify that R(I) and R(O) are equivalence relations and R(S) is a congruence relation.

Conversely, given R(S) define

$S' = \{B_i / B_i \in R(S)\}$, where $B_i$ are blocks of R(S)

$\delta'(B_i, x) = B_j$ for $B_j$ such that $\delta(B_i, x) \subseteq B_j$

and thus $h_S(s) = \{ B_k \in S' / s \in B_k \}$. $h_I$ and $h_O$ are trivial.

As an example, consider the machines of Figs. 1.1 and 1.2.

| | $x_1$ | $x_2$ | $x_3$ | |
|---|---|---|---|---|
| 1 | 5 | 6 | 4 | 2 |
| 2 | 6 | 5 | 4 | 0 |
| 3 | 4 | 4 | 2 | 1 |
| 4 | 3 | 3 | 6 | 2 |
| 5 | 2 | 1 | 3 | 2 |
| 6 | 1 | 2 | 3 | 0 |

Fig. 1.1. Machine $M_{1.1}$

| | 0 | 1 | |
|---|---|---|---|
| A | D | C | $z_1$ |
| B | C | A | $z_2$ |
| C | B | D | $z_1$ |
| D | A | B | $z_1$ |

Fig. 1.2 Machine $M'_{1.1}$

$h_S$: $1 \longrightarrow A$, $2 \longrightarrow A$, $3 \longrightarrow B$, $4 \longrightarrow C$, $5 \longrightarrow D$, $6 \longrightarrow D$

$h_I$: $x_1 \longrightarrow 0$, $x_2 \longrightarrow 0$, $x_3 \longrightarrow 1$

$h_O$: $0 \longrightarrow z_1$, $1 \longrightarrow z_2$, $2 \longrightarrow z_1$

$R(S) = \overline{1,2;3;4;5,6}$ , $R(I) = \overline{x_1, x_2; x_3}$, $R(O) = \overline{0,2;1}$ .

Note: If some $\overline{x} = x_i . x_2 \ldots x_n$, then $h_I(\overline{x}) = h_I(x_1) . h_I(x_2) \ldots . h_I(x_n)$.

## 1.4 Machine Realizations

Definition 1.8: A machine $M = (S, I, O, \delta, \lambda)$ realizes a machine $M' = (S', I', O', \delta', \lambda')$ if and only if there exist three mappings $(\alpha, \beta, \gamma)$ where

$\alpha: S' \longrightarrow$ disjoint subsets of S

$\beta: I' \longrightarrow$ disjoint subsets of I

$\gamma: O' \longrightarrow$ disjoint subsets of O

and these mappings satisfy the following relations:

$$\delta(s,x) \in \alpha(\delta'(s',x')) \text{ for all } s \in \alpha(s'),\ x \in \beta(x') \tag{1}$$

and

$$\lambda(s,x) \in \gamma(\lambda'(s'x')) \text{ for all } s \in \alpha(s'),\ x \in \beta(x') \tag{2}$$

The above model differs somewhat from that given in [ 1 ] but yields the same results provided we assume reduced machines. It avoids certain difficulties and makes the notation more uniform. The idea is that to represent a state of a reduced machine M' we must represent it by at least one state of M and possibly by more than one. Also, two distinct states of M' cannot be represented by the same state of M. Similar considerations apply to input and output coding.

Definition 1.9: A machine M is said to realize the state behaviour of a machine M' if and only if M realizes M' with an assignment $(\alpha, \beta, \gamma)$ such that $\alpha$ maps each state of M' onto a single state of M, and $\alpha$ is one-to-one. [1 ]

Theorem 1.1: A machine M realizes a machine M' if and only if there exists a homomorphism from a submachine of M onto M'.

Proof: We must show that there is a submachine of M which can be mapped homomorphically onto M'. Let

$$S_\sigma = \bigcup \{\alpha(s') \,/\, s' \in S' \text{ of } M'\} \tag{3}$$

$$I_\sigma = \bigcup \{\beta(x') \,/\, x' \in I' \text{ of } M'\} \tag{4}$$

$$O_\sigma = \bigcup \{\gamma(z') \,/\, z' \in O' \text{ of } M'\}. \tag{5}$$

Then $M_\sigma = (S_\sigma, I_\sigma, O_\sigma, \delta_\sigma, \lambda_\sigma)$ is a submachine of M by definition 1.8.

To see that $M_\sigma$ can be mapped homomorphically to M', define

$$h_S(s) = s' \text{ if and only if } s \in \alpha(s') \tag{6}$$

$$h_I(x) = x' \text{ if and only if } x \in \beta(x') \tag{7}$$

$$h_O(z) = z' \text{ if and only if } z \in \gamma(z') . \tag{8}$$

Since, for all $s \in \alpha(s')$, $x \in \beta(x')$

$$\delta_\sigma(s,x) \in \alpha(\delta'(s',x')) ,$$

and

$$\lambda_\sigma(s,x) \in \gamma(\lambda'(s',x')) \text{ for all } s \in \alpha(s'),\ x \in \beta(x') ,$$

we obtain

$$h_S(\delta_\sigma(s,x)) = \delta'(s',x') = \delta'(h_S(s), h_I(x))$$

and

$$h_O(\lambda_\sigma(s,x)) = \lambda'(s',x') = \lambda'(h_S(s), h_I(x)).$$

Conversely, if M is a homomorphic image of $M_\sigma$, then the homomorphism $h = (h_S h_I, h_O)$ defines a realization $(\alpha, \beta, \gamma)$ for M of M' if we let

$$\alpha(s') = \{ s / h_S(s) = s' \} \tag{9}$$

$$\beta(x') = \{ x / h_I(x) = x' \} \tag{10}$$

$$\gamma(z') = \{ z / h_O(z) = z' \} ; \tag{11}$$

and if s is in $\alpha(s')$ then by definition of $\alpha$

$h_S(s) = s'$. Therefore,

$$h_S(\delta_\sigma(s,x)) = \delta'(h_S(s), h_I(x)) = \delta'(s',x') \tag{12}$$

and thus $\delta_\sigma(s,x) \in \alpha(\delta'(s',x'))$.

Also, if we lump into one block all inputs x which map to the same input x', then $\beta$ becomes a one to one mapping from I' to these blocks.

Finally,

$h_O(\lambda_\sigma(s,x)) = \lambda'(s'x')$, and thus

$\lambda\sigma(s,x) \in \gamma(\lambda'(s',x'))$.

## 1.5 Machines in Parallel and in Series

In this section we study machines which can be decomposed into component machines which operate in parallel or in series with one another. Most of this material is based on [ 1 ].

Definition 1.10: The parallel connection of two machines $M_1 = (S_1, I_1, O_1, \delta_1, \lambda_1)$ and $M_2 = (S_2, I_2, O_2, \delta_2, \lambda_2)$ is the machine

$$M = M_1 // M_2 = (S_1 \times S_2, I_1 \times I_2, O_1 \times O_2, \delta, \lambda)$$

with

$$\delta((s_1, s_2), (x_1, x_2)) = (\delta_1(s_1, x_1), \delta_2(s_2, x_2))$$

and

$$\lambda((s_1, s_2), (x_1, x_2)) = (\lambda_1(s_1, x_1), \lambda_2(s_2, x_2)) .$$

Definition 1.11: Given the state machines $M_1 = (S_1, I_1, \delta_1)$ and $M_2 = (S_2, I_2, \delta_2)$ with $I_2 = S_1 \times I_1$ a set of output symbols O and an output function

$$\lambda: S_1 \times S_2 \times I_1 \longrightarrow O$$

then the serial connection of $M_1$ and $M_2$ with the output function $\lambda$ is the machine

$$M = M_1 \ominus M_2 = (S_1 \times S_2, I_1, O, \delta, \lambda)$$

where

$$\delta((s_1, s_2), x) = (\delta_1(s_1, x), \delta_2(s_2, (s_1, x)))$$

and

$$\lambda: S_1 \times S_2 \times I_1 \longrightarrow O .$$

Definition 1.12: The machine $M_1 // M_2$ is a parallel decomposition of M if and only if $M_1 // M_2$ realizes M. Similarly, the machine $M_1 \ominus M_2$ is a serial decomposition of M if and only if $M_1 \ominus M_2$ realizes M.

If $M_1 // M_2$ or $M_1 \ominus M_2$ is a state behaviour realization of M, then we refer to $M_1 // M_2$ as a parallel decomposition of the state

behaviour of M, and $M_1 \ominus\!\!\rightarrow M_2$ as a serial decomposition of the state behaviour of M.

We say that the state behaviour decomposition of M is nontrivial if and only if $M_1$ and $M_2$ have fewer states than M.

Theorem 1.2: A sequential machine M has a nontrivial parallel decomposition of its state behaviour if and only if there exist two nontrivial S.P. partitions $R_1(S)$ and $R_2(S)$ on its states such that $R_1(S) \cdot R_2(S) = 0$. [1]

Theorem 1.3: The sequential machine M has a nontrivial serial decomposition of its state behaviour if and only if there exists a nontrivial S.P. partition R(S) on the states S of M [1].

Definition 1.13: The cross product of a set of states {a} with a set of states {b} is denoted {a} ✕ {b} and defined as the set of states $\{a_i b_j\}$ such that $a_i \in \{a\}$ and $b_j \in \{b\}$. For example: {1,2} ✕ {A,B} = {1A, 1B, 2A, 2B}

Definition 1.14: The direct product of two partitions $R_1(S) \times R_2(S)$ is defined as that partition formed from the set of all products of their blocks.

For example: if $R_1(S_1) = \overline{1,2};\overline{3,4}$

$R_2(S_2) = \overline{A,B};\overline{C,D}$

then $R_1(S_1) \times R_2(S_2) =$

$$\overline{1A,1B,2A,2B};\ \overline{1C,1D,2C,2D};\ \overline{3A,3B,4A,4B};\ \overline{3C,3D,4C,4D}$$

Definition 1.15: A congruence relation R(S) defined on the states of some machine $M = (S_1 \times S_2, I_1 \times I_2, O_1 \times O_2, \delta, \lambda)$ which is the parallel combination of two machines $M_1 = (S_1, I_1, O_1, \delta_1, \lambda_1)$ and $M_2 = (S_2, I_2, O_2, \delta_2, \lambda_2)$ is said to be a free congruence if and only if there exist congruences $R(S_1)$ and $R(S_2)$ defined over $S_1$ and $S_2$ such that

$$R(S) = R(S_1) \times R(S_2) .$$

A homomorphic mapping $h = (h_S, h_I, h_O)$ defined from some machine M to M' where $h_S$ is defined by a free congruence, implies that there exist mappings from the component machines of M to those of M', each mapping defined by one of the congruence relations $R(S_1)$, $R(S_2)$; thus, the mapping from M to M' preserves not only the algebraic structure of M, but that of its component machines as well. Consider the following three machines:

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|---|
| $a_1$ | $a_2$ | $a_2$ | $a_2$ | $a_1$ | $a_3$ |
| $a_2$ | $a_1$ | $a_1$ | $a_1$ | $a_1$ | $a_3$ |
| $a_3$ | $a_4$ | $a_4$ | $a_2$ | $a_3$ | $a_1$ |
| $a_4$ | $a_3$ | $a_3$ | $a_1$ | $a_1$ | $a_1$ |

Fig. 1.3. Machine $M_{1.2}$

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|---|
| $b_1$ | $b_1$ | $b_2$ | $b_2$ | $b_1$ | $b_2$ |
| $b_2$ | $b_2$ | $b_1$ | $b_1$ | $b_1$ | $b_2$ |

Fig. 1.4. Machine $M_{1.3}$

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|---|
| $a_1b_1$ | $a_2b_1$ | $a_2b_2$ | $a_2b_2$ | $a_1b_1$ | $a_3b_2$ |
| $a_1b_2$ | $a_2b_2$ | $a_2b_1$ | $a_2b_1$ | $a_1b_1$ | $a_3b_2$ |
| $a_2b_1$ | $a_1b_1$ | $a_1b_2$ | $a_1b_2$ | $a_1b_1$ | $a_3b_2$ |
| $a_2b_2$ | $a_1b_2$ | $a_1b_1$ | $a_1b_1$ | $a_1b_1$ | $a_3b_2$ |
| $a_3b_1$ | $a_4b_1$ | $a_4b_2$ | $a_2b_2$ | $a_3b_1$ | $a_1b_2$ |
| $a_3b_2$ | $a_4b_2$ | $a_4b_1$ | $a_2b_1$ | $a_3b_1$ | $a_1b_2$ |
| $a_4b_1$ | $a_3b_1$ | $a_3b_2$ | $a_1b_2$ | $a_1b_1$ | $a_1b_2$ |
| $a_4b_2$ | $a_3b_2$ | $a_3b_1$ | $a_1b_1$ | $a_1b_1$ | $a_1b_2$ |

Fig. 1.5. Machine $M_{1.4}$

Now $M_{1.4}$ is constructed as the parallel combination of $M_{1.2}$ and $M_{1.3}$. We would therefore expect to find two partitions with S. P. on the states of $M_{1.4}$, each computed by the component machines. These partitions are:

$$R_1(S) = \overline{a_1b_1, a_1b_2}; \overline{a_2b_1, a_2b_2}; \overline{a_3b_1, a_3b_2}; \overline{a_4b_1, a_4b_2}$$

$$R_2(S) = \overline{a_1b_1, a_2b_1, a_3b_1, a_4b_1}; \overline{a_1b_2, a_2b_2, a_3b_2, a_4b_2} .$$

By defining congruence relations on each of the parallel operating machines, we can generate a free congruence over $M_{1.4}$. For example,

$$R(S_1) = \overline{a_1, a_2}; \overline{a_3}; \overline{a_4} \quad \text{and} \quad R(S_2) = \overline{b}_1; \overline{b}_2 .$$

Then $R(S_1) \times R(S_2) = \overline{a_1b_1, a_1b_2, a_2b_1, a_2b_2}; \overline{a_3b_1}; \overline{a_3b_2}; \overline{a_4b_1}; \overline{a_4b_2}$

which is a free congruence over $M_{1.4}$.

## 1.6. Machines and Circuits

The purpose of this section is to relate the abstract machines to physical circuits.

Definition 1.16: A sequential circuit [3] is a structure consisting of k inputs $v_1, v_2, \ldots, v_k$, m outputs, $z_1, z_2, \ldots, z_m$, s unit delays with outputs $y_1, y_2, \ldots, y_s$, and m+s combinational circuits $f_1, f_2, \ldots, f_s$, $g_1, g_2, \ldots, g_m$ such that

$$y_i(t+1) = f_i(y_1(t), y_2(t), \ldots, y_s(t); v_1(t), v_2(t), \ldots, v_k(t)), \quad t \geq 1 ;$$

$$y_i(1) = y_{i\Lambda}, \text{ a specified starting value, and}$$

$$z_j(t) = g_j(y_1(t), y_2(t), \ldots, y_s(t); v_1(t), v_2(t), \ldots, v_n(t)), t \geq 1 .$$

The inputs, outputs, delays and combinatorial circuits are, in general, multivalued; if they are all binary, the circuit is binary.

Definition 1.17: The machine defined by a circuit [1] is the machine defined in the following way; for input binary variable $v_i$ for $1 \leq i \leq k$, state binary variables $y_j$ for $1 \leq j \leq s$, transition functions $f_j$, and output functions $g_j$ we have

i) $S = \{(y_1, y_2, \ldots, y_s)\} = \{\vec{y}\}$

ii) $I = \{(v_1, v_2, \ldots, v_k)\} = \{\vec{v}\}$

iii) $O = \{(z_1, z_2, \ldots, z_m)\} = \{\vec{z}\}$

iv) $\delta(\vec{y}, \vec{v}) = \{f_j(\vec{y}, \vec{v})\}$

v) $\lambda(\vec{y}, \vec{v}) = \{g_j(\vec{y}, \vec{v})\}$

where $\vec{y} = (y_1, y_2, \ldots, y_n)$ the vector defined by the state variables, etc.

Thus, each circuit defines a unique machine. In this paper only binary circuits are considered. A circuit C, then, is said to realize a machine M' if and only if there exists a homomorphism from a sub-machine M defined by C to the machine M'.

We shall say that a circuit C has some property P if the machine it defines has that property.

From this point on, only state machines will be considered. In this paper, we will not restrict ourselves to machines with logic-free outputs as well as logic-free state realizations, but will consider the more general problem.

## 2. LOGIC-FREE CIRCUITS AND THEIR MACHINES

### 2.1. The General Structure of Logic-Free Circuits

Definition 2.1: A logic-free circuit is a circuit with no combinational circuits; thus, for such a circuit

$$y_i(t+1) = \begin{Bmatrix} 0 \text{ or } 1, \\ v_n(t), \\ y_j(t) \end{Bmatrix}$$

From this definition it is apparent that any delay i in a logic-free circuit can be fed by 1) a constant input of 1 or 0, 2) an input variable $v_n$, or 3) an output of a single delay. Delay i, in turn, can feed 1) nothing, or 2) a finite number of other delays. These circuits are illustrated in Fig. 2.1.

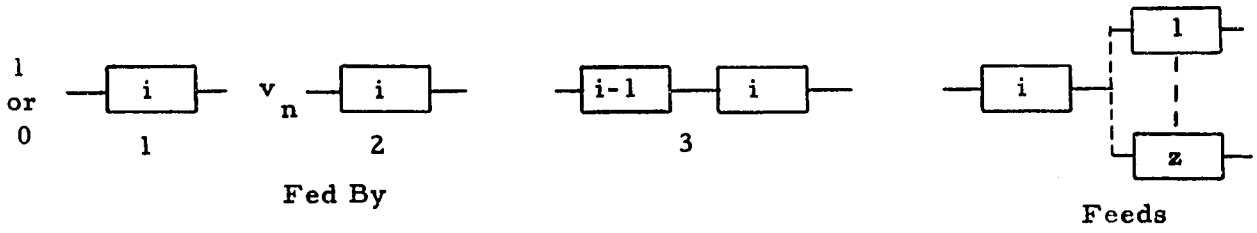



Fig. 2.1. A Delay in a Logic-Free Circuit

The circuits of cases 1) and 2) form tree structures. There are three possibilities for case 3): as we extend the tree in either direction and exhaust all the delays, the tree must eventually be fed by

i) a constant input, in which case, type 1) is formed,
ii) an input variable to yield a type 2) circuit, or
iii) the delays form a complete loop. Such a circuit would be autonomous, or constant input. Its form is shown in Fig. 2.2.

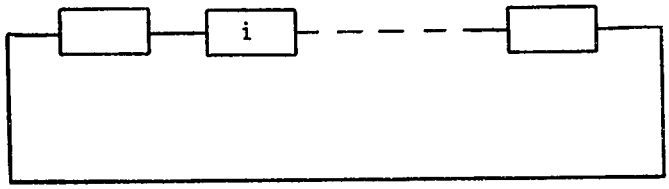



Fig. 2.2. General Form of the Third Type of Circuit.

This circuit consists of a loop in which each delay feeds a finite number of delays.

In summary, there are three types of component circuits in the general logic-free circuit: they are

i) a constant input feedback-free circuit,

ii) an input dependent feedback-free circuit, and

iii) a constant input circuit with feedback.

These are shown in Fig. 2.3.

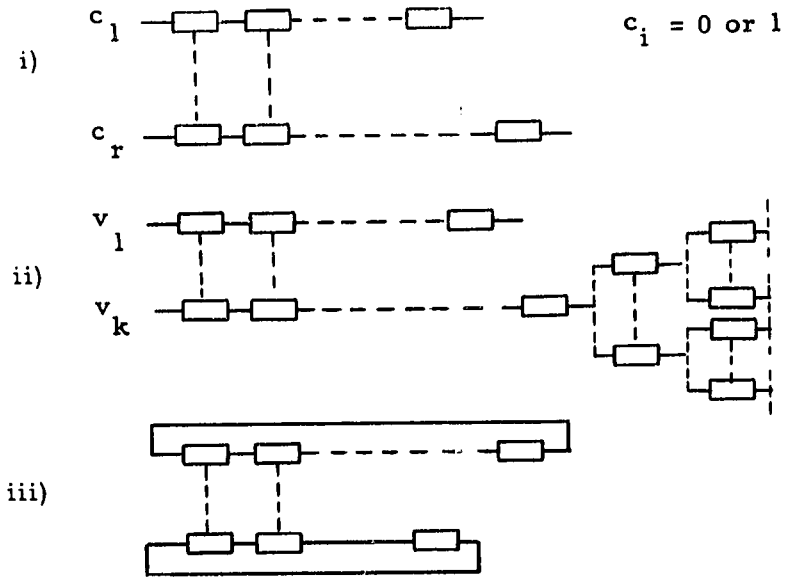



Fig. 2.3. The General Structure of Logic-Free Machines.

## 2.2 Machines Defined by the Component Circuits

Each of the component circuits of the general logic-free circuit defines a particular type of machine. These machines, in turn, are realizable on logic-free circuits. In the following section we will define these machines and outline the techniques with which they can be logic-free realized.

Definition 2.2: A machine will be called a constant input (CI) machine if and only if it has one distinct column in its flow table.

That is, a machine is a CI machine if the next state function is independent of any input variables; mathematically, this implies that, if $M = (S, I, \delta)$ and is CI, then for all $s \in S$ and for all $x_i, x_j \in I$,

$$\delta(s, x_i) = \delta(s, x_j).$$

In the general circuit the constant input component circuits define CI machines.

Definition 2.3: A machine will be called definite if and only if there exists some integer k such that for all $\overline{x} \in I^*$ with $\ell(\overline{x}) \geq k$, and for all $s, s' \in S$

$$\overline{\delta}(s, \overline{x}) = \overline{\delta}(s', \overline{x}). \quad [4]$$

Lemma 2.1: Every feedback-free circuit is definite.

Proof: This has been shown in [5].

Thus, all feedback-free circuits of the general logic-free machine define definite machines.

It follows, then, that the general logic-free circuit defines a machine on whose states it is possible to define a parallel decomposition of the state behaviour into definite, and CI machines. (Theorem 1.2). Now both of these machines have been studied in the literature, and techniques for their realization on logic-free circuits, developed. These

methods will be presented here, with some additions and modifications.

Definite Machines

Definition 2.4: A machine $M = (S, I, \delta)$ is strongly connected if and only for all $s, s' \in S$ there exists some $\bar{x} \in I^*$ such that $\bar{\delta}(s, \bar{x}) = s'$.

Theorem 2.1: The state behaviour of every strongly connected definite machine can be realized on a logic-free circuit.

Proof: This is a result of Theorem 1 in [3]. The assignment is effected with the aid of the input successor tree also given in [3]. It is a graph in the form of a tree and designated G(M). Let S represent the states of the finite machine M. The nodes of the graph correspond to sets of states of M. There is a transition from a set C to a set D under input $x_i$ if D is the set of $x_i$-successor states of C (if state $s_j$ goes to $s_k$ under input $x_i$, then $s_k$ is the $x_i$-successor of $s_j$). The graph is constructed in levels, the zero'th level having one node corresponding to the set of all states. The first level contains all the $x_i$-successors of S and hence has u nodes. The second level contains $u^2$ nodes corresponding to the successors of the sets of states in the first level, etc. If M is a definite machine involving k delays, the state of M reached after applying a sequence of length k + 1 must be independent of the starting state. Hence, if M is definite there must be a level in which each node corresponds to a single state.

The graph is terminated according to the following criteria:

1) just before a set of nodes of any level is regenerated, or
2) a level is reached in which all nodes contain single states.

If G(M) is terminated according to rule 1, the machine is not definite; if it terminates by rule two, however, it is definite [3]. The logic-free assignment is then taken directly from the graph by assigning to each

state the reverse of the sequences required to map to it on the tree. The inputs are assigned arbitrarily. An example is shown in Fig. 2.4.

| $v_1, v_2$ | a | b | c |
|---|---|---|---|
| 1 | 1 | 4 | 5 |
| 2 | 1 | 4 | 5 |
| 3 | 2 | 3 | 6 |
| 4 | 2 | 3 | 6 |
| 5 | 2 | 4 | 5 |
| 6 | 2 | 4 | 5 |

$M_{2.1}$

a ⟶ 00
b ⟶ 01
c ⟶ 11

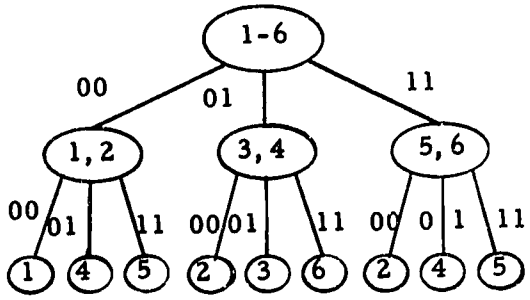



Fig. 2.4. Generation of the Input Successor Tree for Machine $M_{2.1}$.

| | $y_1$ | $y_2$ | $y_3$ | $y_4$ |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 1 | 1 |
| 5 | 1 | 1 | 0 | 0 |
| 5 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 |

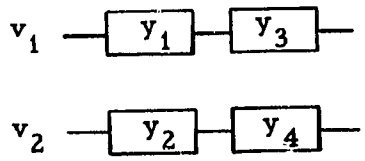



Fig. 2.5. Assignment and Circuit for Machine $M_{2.1}$.

All definite machines are not strongly connected, so that this method will not work in general. Given any definite machine, however, it is always possible to embed it as a submachine in a strongly connected definite machine in the following way:

1) add as many input columns to the given machine as there are states s which do not appear in the final level of the tree G(M);

2) enter into each column one and only one such state s; each state is added to one column;

3) give each column a different input.

It is clear that this larger machine will be both definite and strongly connected, and by definition of a submachine, that the original machine is a submachine of the larger one. Now, realize the larger machine on a logic-free circuit; then, allow only those inputs to the circuit which were coded from inputs to the original machine.

The circuit now realizes the original machine.

Note that adding only one state to an input column is not economical, and often it is possible to add more, yet keep the machine definite. Such is the case with the example of Fig. 2.6.

| | $x_1$ | $x_2$ |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 1 | 2 |
| 3 | 1 | 2 |
| 4 | 1 | 2 |

$M_{2.2}$

Added Input

| | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|
| 1 | 1 | 2 | 3 |
| 2 | 1 | 2 | 3 |
| 3 | 1 | 2 | 4 |
| 4 | 1 | 2 | 4 |

$M'_{2.2}$

Fig. 2.6. Embedding a Definite Machine into a Strongly Connected Definite Machine.

The assignment is now straightforward. In the circuit, the code assigned to $x_3$ is not allowed. We conclude with the following corollary.

Corollary 1: Every definite machine can be realized on a logic-free circuit.

Constant Input Machines

Definition 2.5: An internal state of a machine will be called cyclic if it appears in a cycle; otherwise, it will be called non-cyclic.

Theorem 2.2: Every CI machine can be realized on a logic-free circuit with a single loop.

Proof: This has been proven in [6]. The cyclic states of the machine are realized on what is called a universal circuit, which is actually a single, simple circulating feedback shift register. The non-cyclic states are then coded from transient logic-free registers.

This method is not very economical, particularly if it is not necessary to realize the machine on a single-loop circuit. Certain types of CI machines can be more economically realized, so we make the following distinctions.

Definition 2.6: A machine will be called constant input periodic (CIP) if it is a CI machine with only cyclic states.

The flow graph of such a machine consists of a number of disjoint cycles. Any logic-free circuit which consists only of delays in loops defines a CIP machine, and conversely, any CIP machine can be realized on such a circuit. However, there is no algorithm available for an assignment which guarantees the minimum number of delays.

Definition 2.7: A machine will be called constant input tree-type (CIT) if it is a CI machine with one and only one cyclic state.

The constant input feedback free circuit of the general logic-free machine defines a CIT machine.

The logic-free realization of CIT machines is much simpler than that of CIP machines or CI machines in general. In the following we will develop an algorithm which will lead to an economical realization for such machines. First, however, we will review some of the theory of trees.

The state graph shown in Fig. 2.7 is a tree. A tree contains three kinds of states: a primary state which has no incoming branches or no predecessors; a root state which has itself as next state; and a secondary state which is neither a primary state nor a root state. Both the primary and secondary states are non-cyclic states. The root state is a cyclic state.

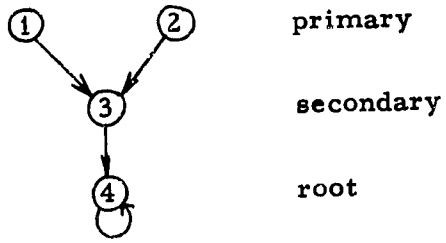



Fig. 2.7. A Tree.

Definition 2.7: A tree is regular if all states but the primary states have the same number of incoming branches. [7].

The root state can be reached from any other state of the tree. The minimum number of transitions for a state to reach the root is called the level of the state; thus, the level of the root state is zero.

Definition 2.8: The tree set of a tree is a set of integers $(h_0, h_1, \ldots, h_L)$ where $h_i$ is the number of states of the level i, and L is the highest level in the tree. L is called the height of the tree. [7].

The tree set describes the regular tree up to isomorphism. Every state except the terminal state in a regular tree has $h_i + 1$ incoming

branches. The number of primary states $t_i$ on level i is

$$t_i = h_i - (h_{i+1}/(h_1+1), \quad 0 < i < L,$$
$$t_0 = 0,$$
$$t_L = h_L.$$

Constant Input Feedback-Free Circuits and Their Tree-Graphs.

The circuit structure we will use to implement transient machines is constructed of logic-free registers and called a register circuit. One such circuit is illustrated in Fig. 2.8. The circuit is divided into levels, each level containing a number of delays: the first level contains only those delays which do not feed other delays; the second level contains those delays which feed first level delays, the third those which feed second level delays etc. Each delay feeds at most one other delay.

Definition 2.9: The delay set of a register circuit is a set of integers $< d_1, d_2, \ldots, d_L >$ where $d_i$ is the number of delays of level i and L is the highest level of the circuit. L is called the length of the circuit.

A register circuit is determined up to isomorphism by its delay set.

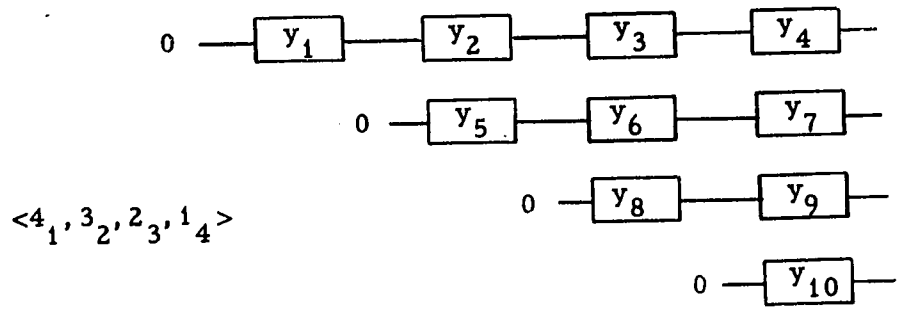



Fig. 2.8. A Register Circuit and its Delay Set

Tree Graphs of Register Circuits

There is a definite relationship between the delay set of a register circuit and the tree set of its state graph; it is shown in the following Lemma:

Lemma 2.2: Given the delay set $\langle d_1, d_2, \ldots, d_L \rangle$ of some transient circuit; assume that each delay carries a p-valued signal. The tree set of its state graph is given by the formula;

$$h_i = (p^{d_i} - 1) \prod_{k=(d_i-1)}^{1} (p^{d_k}).$$

Proof: In any tree set, $h_0 = p^0 = 1$ by definition. Now, the number of states that reach the root state (or the all zero state) in one transition is $p^{d_1} - 1$. Similarly, the number of states which clear to the root in two transitions is equal to the number of states which have at least one non-zero entry in the second level delays but no entry in any higher level delays, less the all zero tuple. This is given by $(p^{d_2} - 1)\, p^{d_1}$. By induction the general formula becomes

$$h_i = (p^{d_i} - 1) \prod_{k=d_i-1}^{1} (p^{d_k}). \qquad (1)$$

We can further characterize the state graphs of these machines by claiming;

Lemma 2.3: The state graph of any register circuit is a regular tree.

Proof: The proof follows from [7], Theorems 1 and 2.

The method we use to synthesize tree type machines is to determine, by inspection, the smallest regular tree of which the tree of our machine is a part (or a subtree)* and then apply our formula (1) to

* T' is a subtree of T provided the machine which defines T' is a sub-machine of the machine which defines T.

derive the delay set of the smallest circuit which will realize the regular tree. The procedure follows:

1) determine the largest number of incoming branches to any single state of the tree; call this number a.
2) add to each secondary state and the root state the necessary number of predecessors so that each secondary state and the root state have the same number of incoming branches, a.
3) specify the tree set of this regular tree.

$$(1, h_1, h_2, \ldots, h_L)$$

From the equation of Lemma 1.4 we can derive design formulas for any transient machine, as

$$h_0 = 1$$

$$h_1 = p^{d_1} - 1 \text{ and thus } p^{d_1} = h_1 + 1$$

thus, $$d_1 = [\log_p(h_1 + 1)] \text{ where } [x] \text{ implies the smallest positive integer} \geq x$$

Similarly $$d_2 = [\log_p(h_2 + p^{d_1})] - d_1$$

In general, our formula is

$$d_i = [\log_p(h_i + \prod_{k=i-1}^{1} (p^{d_k})] - \sum_{k=i-1}^{1} d_k \cdots$$

For example, consider the machine of Fig. 2,9, without the starred states

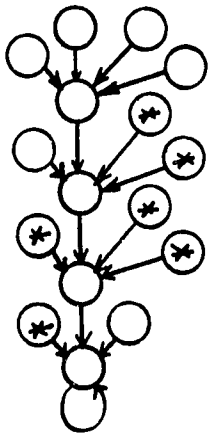


Fig. 2.9. A Tree Type Machine $M_{2,3}$.

The tree set of the smallest regular tree of which this is a subtree is

$$(1_o, 3_1, 4_2, 4_3, 4_4)$$

The delay set of the smallest machine of which the regular tree is a subtree is:

$$d_1 = [\log_2(3)] = 2$$

$$d_2 = [\log_2(4+4)] - 2 = 1$$

$$d_3 = [\log_2(4+4)] - 2 = 1$$

$$d_4 = \log_2(4+4) - 2 = 1$$

The circuit is shown in Fig. 2.10.

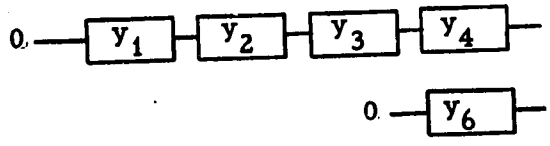



Fig. 2.10. The Register Circuit of $M_{2,3}$.

This brings us to the main theorem of the section.

Theorem 2.3: The state behaviour of any CIT machine can be realized without logic on a register circuit.

## 2.3 Image Machines and Submachines of Definite and CI Machines

Up to this point, given a definite or CI machine we can realize it on a logic-free circuit; if the CI machine is the special case of a CIP, then the realization is somewhat simpler. However, recalling that a CIT machine is realized by another if and only if it is an image machine of a submachine, the most obvious next step is to characterize all those machines which can be realized by definite and CI machines. Such is the purpose of this section.

Theorem 2.4: The submachines of definite or CI machines are respectively definite or CI machines.

Proof: It is easily verified that each submachine enjoys the necessary properties to characterize it as definite or CI.

Theorem 2.5: The homomorphic image machines of definite or CI machines are respectively definite or CI machines.

Proof:

1) Definite Machines: Assume there exists a mapping $h = (h_S, h_I)$ from some definite machine $M = (D, I, \delta)$ to some machine $M' = (S, I', \delta')$; then, since M is definite there exists some integer k such that for all $\bar{x} \in I^*$ with $\ell(\bar{x}) \geq k_1$,

$\bar{\delta}(d, \bar{x}) = \bar{\delta}(d', \bar{x})$ for all $d, d' \in D$. This means $h_S(\bar{\delta}(d, \bar{x})) = h_S(\bar{\delta}(d', \bar{x}))$ for all $d, d' \in D$ which implies that $\bar{\delta}'(h_S(d), h_I(\bar{x})) = \bar{\delta}'(h_S(d'), h_I(\bar{x}))$ for all $d, d' \in D$;

but h is an onto mapping, and thus,

$$\overline{\delta}'(s, h_I(\overline{x})) = \overline{\delta}'(s', h_I(\overline{x})) \text{ for all } s, s' \in S,$$

which, by definition, means that M' is a definite machine.

2) CI Machines: Assume there exists a mapping $h = (h_S, h_I)$ from some CI machine $M = (A, I, \delta)$ to some machine $M' = (A', I', \delta')$; then, since M is CI; for all $a \in A$, for all $x_i, x_j \in I$

$$\delta(a, x_i) = \delta(a, x_j) \text{ which means that}$$

$$h_S(\delta(a, x_i)) = h_S(\delta(a, x_j)) \text{ and thus,}$$

$$\delta'(h_S(a), h_I(x_i)) = \delta'(h_S(a), h_I(x_j)) \text{ under the same conditions;}$$

but h is onto, so that

$$\delta'(a', h_I(x_i)) = \delta'(a', h_I(x_j)) \text{ for all } a' \in A',\ h_I(x_i), h_I(x_j) \in I'.$$

By definition this implies that M' is a CI machine.

Corollary 1: A machine M can be realized by a definite machine if and only if it is itself a definite machine, and by a CI machine if and only if it is a CI machine.

There are a few more important properties of image machines which will be required later. These are:

Lemma 2.4: Every submachine and image machine of a CIP machine is a CIP machine.

Proof: Since a CIP machine consists of disjoint cycles each of which are strongly connected, it is clear that any submachine must also be a CIP machine.

Assume $M = (S, I, \delta)$ is CIP, and it maps to $M' = (S', I', \delta')$ under $h = (h_S, h_I)$. Then, for all $s \in S$ there must exist some $\overline{x} \in I^*$ such that $\overline{\delta}(s, \overline{x}) = s$, which implies that

$h_S(\bar{\delta}(s, \bar{x})) = h_S(s)$, and thus $\overline{\delta'}(h_S(s), h_I(\bar{x})) = h_S(s)$;

but h is onto, so that for all $s' \in S'$ there must exist some $\bar{x}' \in I'$ such that $\delta'(s', \bar{x}') = s'$.

But this means that all states of S' are cyclic, and that M' must be CIP.

Lemma 2.5: Every image machine $M' = (S', I', \delta',)$ of a strongly connected machine $M = (S, I, \delta)$ is a strongly connected machine.

Proof: Let $h = (h_S, h_I)$ be the mapping from M to M': then, since M is strongly connected, for all $s_1, s_2 \in S$ there exists some $\bar{x} \in I^*$ such that

$$\bar{\delta}(s_1, \bar{x}) = s_2 \text{ which means that}$$

$$h_S(\bar{\delta}(s_1, \bar{x}) = h_S(s_2) \text{ and then}$$

$$(h_S(s_1), h_I(\bar{x}) = h_S(s_2).$$

But h is onto, so that, for all $s'_1, s'_2 \in S'$, there exists some $h_I(\bar{x}) \in I^*$ such that $\bar{\delta}'(s'_1, h_I(\bar{x})) = s'_2$. By definition, M' must be strongly connected.

## 2.4 Image Machines and Submachines of the General Circuit

Image machines and submachines of definite and CI machines have been characterized. What of machines which consist of the parallel combination of such machines - the type of machine defined by the general logic-free circuit? We will now show that the submachines of such circuits also consist of the parallel combination of definite and CI machines, but that the image machines need not. The result is one of the central theorems of the paper.

Theorem 2.6: Given any machine $M = (S, I, \delta) = M_1 // M_2$ which consists of the parallel combination of two machines, $M_1$ and $M_2$; then any sub-

machine $M_\sigma$ of M consists of the parallel combination of submachines of $M_1$ and $M_2$.

Proof: That $M = M_1 // M_2$, implies that there exists two congruence relations on S, call them $R_1(S)$ and $R_2(S)$, computed by $M_1$ and $M_2$ respectively, such that $R_1(S) \cdot R_2(S) = 0$. Let $M_\sigma = (S_\sigma, I_\sigma, \delta_\sigma)$ be any submachine of M. Define $R_1(S_\sigma)$ and $R_2(S_\sigma)$ as $R_1(S)$ and $R_2(S)$ with those states which are not contained in $S_\sigma$, removed. Clearly, removing these states does not upset the property that no two states in the same block of one partition belong to the same block of the other, and so $R_1(S_\sigma) \cdot R_2(S_\sigma) = 0$. Further, since $I_\sigma$ is a subset of I, the two partitions defined over the submachine must have S.P.. From Theorem 1.2 this implies that $M_\sigma$ has a parallel decomposition on its state behaviour. Now the two partitions defined over the submachine either have the same number of blocks as the original partitions or not. If so, then the component parallel machines are the same as the original component machines. If not, then obviously they are submachines of the originals.

Since every submachine of a definite machine is a definite machine, and similarly every submachine of a CI machine is a CI machine, we can conclude with the following theorem.

Theorem 2.7: A machine can be realized without logic if and only if it is a homomorphic image of some machine which consists of the parallel combination of a definite machine with a CI machine.

Note: There exists some machine $M = M_1 // M_2$ where $M_1$ is definite and $M_2$ is CI, which has an image machine for which there does not exists a parallel decomposition of its state behaviour into a definite machine and a CI machine. This we will show with the following example:

| | 0 | 1 |
|---|---|---|
| A | G | A |
| B | E | C |
| C | E | C |
| D | G | A |
| E | H | B |
| F | F | D |
| G | F | D |
| H | H | B |

$M_{2.4}$

| | 0 | 1 |
|---|---|---|
| 1 | 5 | 1 |
| 2 | 4 | 2 |
| 3 | 5 | 1 |
| 4 | 6 | 2 |
| 5 | 5 | 3 |
| 6 | 6 | 2 |

$M_{2.5}$

Fig. 2.11. Flow Tables for Machines $M_{2.4}$ and $M_{2.5}$.

The machine $M_{2.4}$ has the two partitions

$R_i = \overline{A, D, F, G}; \overline{B, C, E, H}$, the input independent partition

$R_d = \overline{A, C}; \overline{B, D}; \overline{F, H}; \overline{E, G}$, the partition computed by the definite machine.

Now the mapping from $M_{2.4}$ to $M_{2.5}$ is

$h_S$: A $\longrightarrow$ 1, B $\longrightarrow$ 2, C $\longrightarrow$ 2, D $\longrightarrow$ 3, E $\longrightarrow$ 4, F $\longrightarrow$ 5, G $\longrightarrow$ 5, H $\longrightarrow$ 6

$h_I$ is trivial.

However, by generating all the partitions with S. P. over $M_{2.5}$ it can be verified that the desired parallel decomposition does not exist.

## SUMMARY

From the description of the general circuit structure of logic-free machines, it was shown that the machine which such a circuit defines can be decomposed on its state behaviour into the parallel combination of a definite and a C.I. machine. Further, any submachine of such a circuit is similarly decomposable. Since each of these component machines are logic-free realizable by previously developed methods, it follows, then, that a machine is logic-free realizable if and only if it is a homomorphic image of some machine which consists of the parallel combination of a definite and a C.I. machine.

In the following sections, such image machines will be studied.

## 3. CYCLIC MACHINES

By Theorem 2.7, a machine M' can be realized on a logic-free circuit C if and only if it is a homomorphic image of some machine $M = M_1 // M_2$, with $M_1$ a definite machine and $M_2$ a CI machine. However, unlike M, machine M' need not be decomposable on its state behaviour into two such component machines. The obvious conclusion, then, is that there are two types of logic-free realizable machines: 1) those with the parallel decomposition defined on their state behaviour, and 2) those without it. We will consider both types in this section.

Because non-cyclic states in a machine create complications and are difficult to handle, we will confine this section to machines which have only cyclic states. We call them cyclic machines. It is clear that if a machine $M = M_1 // M_2$ is cyclic, then both $M_1$ and $M_2$ are cyclic. Now, any definite cyclic machine is strongly connected, and any cyclic CI machine is a CIP machine by definition. Therefore, in this section we will be concerned with characterizing those machines which are images of machines composed of a definite strongly connected machine and a CIP machine in parallel.

### 3.1: Decomposable Cyclic Machines

Hartmanis and Stearns [1] have shown that those partitions computed by the set of all CI machines which can be used as 'front' machines in a series decomposition on the state behaviour of some machine M, form a lattice. Therefore, there exists a unique smallest partition $R_{ci}(S)$ computed by the largest such CI machine.

Similarly, they have shown that there exists a unique smallest partition $R_{ff}(S)$ computed by the largest feedback-free front machine.

It follows, therefore, that to determine if the state behaviour of M is realizable as a parallel combination of a CI machine and a

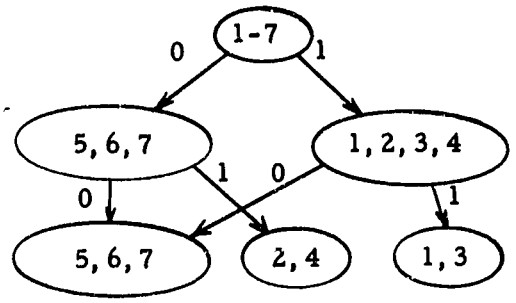
definite machine, it suffices to test whether $R_{ci}(S) \cdot R_{ff}(S) = 0$.

Hartmanis and Stearns [1 ] give methods for generating both of these partitions. $R_{ci}(S)$ is determined for some cyclic machine M with n states as follows: first, make a block of all the states of the machine that are contained in the same row of the flow table, and then construct a partition $R_i(S)$ by combining those blocks with common elements (chain connecting). Now, find the smallest partition with S. P. which contains $R_i(S)$. This will be $R_{ci}(S)$: $R_{ff}(S)$ is constructed as follows: generate the input successor tree G(M) of M, only this time chain connect the nodes on each level before going to the next level. The partition generated by the nodes of the nth level is $R_{ff}(S)$.

Example:

| | $x_1$ | $x_2$ |
|---|---|---|
| 1 | 6 | 1 |
| 2 | 5 | 3 |
| 3 | 5 | 3 |
| 4 | 6 | 1 |
| 5 | 7 | 2 |
| 6 | 6 | 4 |
| 7 | 7 | 2 |

$M_{3.1}$




Generation of $R_{ff}(S)$

$$R_i(S) = \overline{1,4,6};\ \overline{3,5};\overline{2,7}$$

$$R_{ci}(S) = \overline{1,4,6};\overline{2,3,5,7}$$

$$R_{ff}(S) = \overline{1,3};\overline{2,4};\overline{5,6,7}$$

Fig. 3.1. Generation of the Partitions for Machine $M_{3.1}$.

In this example, $R_{ff}(S) \cdot R_{ci}(S) \neq 0$. Consider $M_{3.2}$ of Fig. 3.2.

| | $x_1$ | $x_2$ |
|---|---|---|
| 1 | 7 | 1 |
| 2 | 5 | 3 |
| 3 | 5 | 3 |
| 4 | 7 | 1 |
| 5 | 8 | 2 |
| 6 | 6 | 4 |
| 7 | 6 | 4 |
| 8 | 8 | 2 |

$R_{ci}(S) = \overline{1,4,5,7;2,3,5,8}$

$R_{ff}(S) = \overline{1,3};\overline{2,4};\overline{5,7};\overline{6,8}$

Fig. 3.2. Flow Table and Partitions of Machine $M_{3.2}$

Since $R_{ci}(S) \cdot R_{ff}(S) = 0$, the machine $M_{3.2}$ can be realized without logic, each component machine independently of the other, using the methods of section 2.

## 3.2 Characterization of Cyclic Image Machines

All we know of the second class of machines is that they are images of some $M = M_1 // M_2$ where $M_1$ is definite strongly connected and $M_2$ is CIP. Now, consider an arbitrary machine $M = (S, I, \delta) = M_1 // M_2$ and an arbitrary image of it $M'' = (U, I', \delta'')$ induced by some homomorphism $h = (h_S, h_I)$. Since $M''$ is completely characterized by M and h, and M itself is well defined, a study of $M''$ can be made indirectly by characterizing h. Now $h_I$ is straight forward, so that $h_S$ alone needs inspection. It, in turn, is defined by some congruence relation R(S) on the states S of M. Thus, the problem of characterizing $M''$ is reduced to that of characterizing all congruences on the states of M, and thus the mappings $h_S$ that they define.

This characterization will be effected in the following way: all mappings $h = (h_S, h_I)$ from M to M'' will be decomposed into two mappings $h_F = (h_{SF}, h_I)$ from M to some M', and $h_K = (h_T, h_I)$ from M' to M''. $h_{FS}$ will be defined by a free congruence relation F(S) on S, and $h_T$ by a congruence K(T) on the states T of M'. Since F(S) is a free congruence, M' will consists of the parallel combination of a definite strongly connected machine and a CIP machine, so that $R_{ci}(T) \cdot R_{ff}(T) = 0$. It is then logic-free realizable using methods developed previously. However, the mapping $h_K$ may destroy this decomposition property. But, $h_K$ will have redeeming qualities which will prove that the tree G(M) can generate its inverse, and so reclaim the logic-free realizable machine M' as a projection of M''.

The Congruence Relation F(S) and its Mapping $h_{FS}$

Definition 3.1: Let some machine $M = (S, I, \delta) = M_1 // M_2$ where $M_i = (S_i, I, \delta_i)$ and $M_2 = (S_2, I, \delta_2)$; let R(S) be some congruence relation on S; define a relation $R(S_i)$ on $M_i$ for i = 1, 2 as follows:

$$\text{for all } s_\ell, s_k \in S_i, \quad s_\ell R(S_i) s_k \longleftrightarrow (s_\ell, u) R(S) (s_k, u) \text{ for all } u \in S_j$$

where $j = 2$ if $i = 1$, and $j = 1$ if $i = 2$.

Lemma 3.1: $R(S_i)$ is an equivalence relation on $S_i$.

Proof: This is easily verified.

Theorem 3.1: If every state of $M_j$ has a predecessor, then $R(S_i)$ is a congruence relation on $S_i$.

Proof: For all $s_j, s_k \in S_i$, $s_j R(S_i) s_k \longrightarrow (s_j, u) R(S) (s_k, u)$ for all $u \in S$. But R(S) is a congruence relation, and thus,

$(\delta((s_j, u), x))R(S)(\delta((s_k, u), x))$ for all $u \in S_j$, $x \in I$, or

$(\delta_i(s_j, x), \delta_j(u, x))R(S)(\delta_i(s_k, x)\delta_j(u, x))$ $u \in S_j$, $x \in I$.

Since every state of $M_j$ appears as a next state,

$$\delta_j(u, x) = S_j$$

for all $u \in S_j$

all $x \in I$

Thus, $(\delta_i(s_j, x), u)$ $R(S)$ $(\delta_i(s_k, x), u)$ for all $u \in S_j$, $x \in I$ which, by definition of $R(S_i)$, implies

$$(\delta_i(s_i, x))\ R(S_i)\ (\delta_i(s_k, x)) \text{ for all } x \in I.$$

Hence, $R(S_i)$ is a congruence relation.

Corollary 1: For any sequential machine $M = (S, I, \delta) = M_1 // M_2$, where $M_1 = (D, I_1, \delta_1)$ is definite strongly connected, and $M_2 = (A, I_2, \delta_2)$ is CIP, $R(D)$ and $R(A)$ as given in Definition 3.1 are congruence relations.

Proof: It is simple to verify that the conditions of Theorem 3.1 hold for strongly connected definite and CIP machines.

Defintion 3.2: Under the conditions of Corollary 1, define a relation $F(S)$ on the states S of M as follows:

$$F(S) = R(D) \times R(A): \text{ in other words,}$$

$$(d_1, a)\ F(S)\ (d_2, a) \longleftrightarrow d_1\ R(D)\ d_2, \text{ and}$$

$$(d, a_1)\ F(S)\ (d, a_2) \longleftrightarrow a_1\ R(A)\ a_2$$

and no other states are related by $F(S)$.

Lemma 3.2: $F(S)$ is a congruence relation, and $F(S) \subseteq R(S)$.

Proof: By straightforward application of the definitions it can be verified that $F(S)$ is a congruence relation. To prove that $F(S) \subseteq R(S)$, suppose that

$$(d_1, a_i)\ F(S)(d_2, a_i) \longrightarrow d_1\ R(D)\ d_2 \rightarrow (d_1, a)\ R(S)\ (d_2, a)$$

for all $a \in A$, and in particular, $(d_1, a_i)\ R(S)\ (d_2, a_i)$.

Similarly,

$$(d, a_1)\ F(S)(d, a_2) \longrightarrow (d, a_1)\ R(S)\ (d, a_2).$$

Hence, $F(S) \subseteq R(S)$.

From Lemma 3.2 and by Definitions 1.15 and 3.2, F(S) is a free congruence.

Now, let us assume that there is some arbitrary mapping $h = (h_S, h_I)$ from $M = M_1 // M_2$, where $M_1$ is definite strongly connected, and $M_2$ is a CIP machine, to $M'' = (U, I', \delta'')$. Call the congruence relation that defines $h_S$, R(S). From the above development, it is possible to define another mapping, this time from M to some $M' = (T, I', \delta')$ by $h_F = (h_{FS}, h_I)$ where $h_{FS}$ is defined by the free congruence F(S), and $h_I$ is the same as in h. This would imply two properties:

i) M' is larger than M'', as $F(S) \subseteq R(S)$.

ii) $M' = M'_1 // M'_2$ where $M'_1$ is an image machine of $M_1$, and $M'_2$ is an image machine of $M_2$. But, by Theorem 2.5, and Lemmas 2.4 and 2.5, $M'_1$ must be definite strongly connected as $M_1$ is, and $M'_2$ must be a CIP machine because $M_2$ is. Thus, $R_{ci}(T) \cdot R_{ff}(T) = 0$, and the machine is decomposable on its state behaviour.

The machine M' is still not the general case. Since $F(S) \subseteq R(S)$ and although M' has the same input alphabet I' as M'', the two machines are not necessarily the same. In order to make M' the same as M'', it is necessary that F(S) = R(S). This can be done by merging those blocks of F(S) with states related by R(S). In terms of the machines, this means that those states of M' which have inverses under $h_F$ related by R(S) must be merged. This is done by the relation K(T), defined as follows.

Definition 3.3: Let $M' = (T, I', \delta')$ be the image machine of M induced by $h = (h_{SF}, h_I)$. Define a relation K(T) on T as follows:

$$t\ K(T)\ t' \longleftrightarrow \text{there exist } s, s' \in S \text{ such that}$$
$$h_F(s) = t,\ h_F(s') = t',\ \text{and } s\ R(S)\ s',\ \text{for } t, t' \in T$$

Lemma 3.3: K(T) is a congruence relation on M'.

Proof: This is easily verified.

Thus, by definition of K(T), $h_T$ maps T to U. If we let $h_{I'}$ be the identity map, then $h_K$ maps M' to M''.

In summary, we have shown that any mapping $h = (h_S, h_I)$ from some machine $M = M_1 // M_2$ where $M_1$ is definite strongly connected, and $M_2$ is a CIP, to some machine $M'' = (U, I', \delta'')$ can be decomposed into two mappings $h_F = (h_{FS}, h_I)$ and $h_K = (h_T, h_{I'})$ from M to $M' = (T, I', \delta')$, and M' to M'' respectively. Since the relation which defines $h_{FS}$ is a free congruence, M' is also decomposable on its state behaviour into definite strongly connected and CIP machines. It is therefore logic-free realizable by methods developed in earlier sections.

The problem still remains to characterize M''. Now, however, we can do this by characterizing $h_K$ or K(T). This is done in the following theorem.

Theorem 3.2: Let $h_K = (h_T, h_{I'})$ be defined from $M' = M_1' // M_2'$ where, as proven, $M_1' = (D', I_1', \delta_1')$ is definite strongly connected, and $M_2' = (A', I_2', \delta_2')$ is CIP. $h_T$ is defined by K(T). Then, K(T) cannot merge two different states of M' with identical definite components.

Proof: Assume $(d_1', a_1')\ K(T)\ (d_1', a_2')$ where $a_1' \neq a_2'$. Now, $d_1', a_1', a_2'$ are images of at least three states of $M_1$ and $M_2$, call them $d_1, a_1, a_2$ respectively. By definition of K(T), then,

$(d_1, a_1)\ R(S)\ (d_1, a_2)$ which means that

$$(\bar{\delta}((d_1, a_1), \bar{x}))\ R(S)\ (\bar{\delta}((d_1, a_2), \bar{x}) \text{ for all } \bar{x} \in I^* \tag{1}$$

Let $\ell(\bar{x}) = p$, where p is a multiple of all cycles of the CIP component machine, and is greater than or equal to the integer k of the definite machine. Then, (1) implies

$$(\bar{\delta}_1(d_1, \bar{x}),\ \bar{\delta}_2(a_1, \bar{x}))\ R(S)\ (\bar{\delta}_1(d_1, \bar{x}),\ \bar{\delta}_2(a_2, \bar{x})) \tag{2}$$

but, $\bar{\delta}_1(a, \bar{x}) = a$ for all $a \in A$ and all $\bar{x}$ with $\ell(\bar{x}) = p$.

Thus, (2) becomes

$$(\bar{\delta}_1(d_1, \bar{x}), a_1)\ R(S)\ (\bar{\delta}_1(d_1, \bar{x}), a_2) \text{ for all } \bar{x} \text{ with } \ell(\bar{x}) = p$$

but

$$\bar{\delta}_2(d, \bar{x}) = D\ ,$$

$$\text{all } \bar{x} \in I^* \text{ with } \ell(\bar{x}) = p$$

and thus $(d, a_1)\ R(S)\ (d, a_2)$ for all $d \in D$, which implies that $a_1\ R(D)\ a_2$, and that $(d, a_1)\ F(S)\ (d, a_2)$.

But, if $(d_1, a_1)\ F(S)\ (d_1, a_2) \longrightarrow (d_1', a_1') = (d_1', a_2')$

which contradicts our original assumption, and proves the theorem.

## K(T) and the Input Successor Tree

Theorem 3.2 proved that only those states of M' which have different definite components can be related by K(T). Assume the input successor tree were applied to some such M'. Its final level yields a set of nodes such that each is occupied by a set of states with common definite components, and each definite component is represented by at least one such node. Now, assume that two states of M' were merged - say state $(a_1', d_1')$ and $(a_2', d_2')$ - to form state q of a new machine M''.

Applying the input successor tree to M'', it is clear that the state q will appear in two nodes of the final level - those which contained states with $d'_1$ components, and those which contained states with $d'_2$ components. This condition is easily recognized, and it is a simple matter to split all such states that appear in more than one node. Thus, the inverse mapping can be determined, and the machine M' generated. This will be a projection of M'' which, as shown, is logic-free realizable. The two partitions $R_{ff}(S)$ and $R_{ci}(S)$ are then generated, and each component machine realized. An example is given in Fig. 3.3.

Example:

| | 0 | 1 |
|---|---|---|
| 1 | 5 | 1 |
| 2 | 4 | 2 |
| 3 | 5 | 1 |
| 4 | 6 | 2 |
| 5 | 5 | 3 |
| 6 | 6 | 2 |

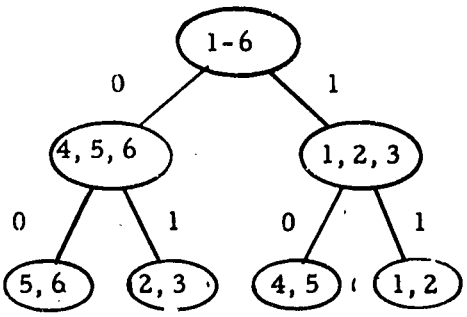



Fig. 3.3. Flow Table and Input Successor Tree for Machine $M_{3.3}$

States 5 and 2 appear in two nodes, and so must be split to, say, $5_a, 5_b$ and $2_a, 2_b$. The resultant projection $P(M_{3.3})$ is given in Fig. 3.4. (Note that where a complete node appears twice, it is not necessary to split all states in it).

| | 0 | 1 |
|---|---|---|
| 1 | $5_b$ | 1 |
| $2_a$ | 4 | $2_b$ |
| $2_b$ | 4 | $2_b$ |
| 3 | $5_b$ | 1 |
| 4 | 6 | $2_a$ |
| $5_a$ | $5_a$ | 3 |
| $5_b$ | $5_a$ | 3 |
| 6 | 6 | $2_a$ |

$$R_i = \overline{1, 5_b}; \overline{4, 2_b}; \overline{6, 2_a}; \overline{3, 5_a}$$

$$R_{ci} = \overline{1, 3, 5_a, 5_b}; \overline{2_a, 2_b, 4, 6}$$

$$R_{ff} = \overline{1, 2_b}; \overline{2_a, 3}; \overline{4, 5_b}; \overline{5_a, 6}$$

Fig. 3.4. Flow Table for $P(M_{3.3})$ and Partitions.

The logic-free realization for this machine is given in Fig. 3.5.

| $y_1$ | $y_2$ | $y_3$ | v | 0 | 1 |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 001 | 011 |
| 1 | 1 | 0 | $2_a$ | 101 | 111 |
| 1 | 1 | 1 | $2_b$ | 101 | 111 |
| 0 | 1 | 0 | 3 | 001 | 011 |
| 1 | 0 | 1 | 4 | 100 | 110 |
| 0 | 0 | 0 | $5_a$ | 000 | 010 |
| 0 | 0 | 1 | $5_b$ | 000 | 010 |
| 1 | 0 | 0 | 6 | 100 | 110 |

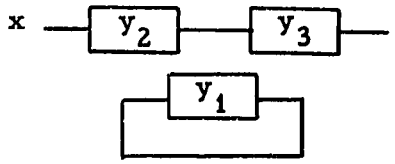



Fig. 3.5. Realization of Machine $M_{3.2}$.

This leads us to our final theorem of this section.

Theorem 3.3; The necessary and sufficient condition for the logic-free realization of the state behaviour of any cyclic machine is that the product of the smallest input-independent partition with the definite partition, on the projection machine which is generated by the input successor tree, is the trivial partition.

## 4. LOGIC-FREE MACHINES IN GENERAL

One class of machines with non-cyclic states, the CIT machine, has been studied in detail. The logic-free realization of cyclic machines was also solved. In this section, we will combine the techniques developed for the above cases to solve the general problem.

It will be shown that any machine which is logic-free realizable must have a projection which can be decomposed on its state behaviour into the parallel combination of a CIT machine with a cyclic logic-free realizable machine. Each of these are then realizable individually on logic-free circuits.

As in the case of cyclic machines, then, we will define two classes of logic-free realizable machines: 1) those with the parallel decomposition on their state behaviour into CIT and cyclic machines, and 2) those without it.

### 4.1: Decomposable Logic-Free Machines

The set of partitions generated by the CIT machine form a sublattice of the set generated by the CI machines(easily confirmed). The smallest such partition $R_{cit}(S)$ can be selected from this set. However, it can also be generated from $R_{ci}(S)$ in the following way: generate $R_i(S)$ (If non-cyclic states are present, $R_i$ as previously defined will not be a partition, for the primary states do not appear in the next state columns. In this event, let each primary state be a different block of the partition.). Generate $R_{ci}(S)$, and then merge all blocks which contain cyclic states into one. The resultant partition is $R_{cit}(S)$.

Generating the partition computed by the cyclic submachine is not as simple, however. Assume we are given a sequential machine $M = (S, I, \delta)$. Let $S = S_c \cup S_{nc}$ where $S_c$ is the set of cyclic states

and $S_{nc}$ the set of non-cyclic states. Assume $M_c = (S_c, I, \delta_c)$ is a submachine (in the case of logic-free realizable machines, it will be shown that it always is) where $\delta_c$ is $\delta$ restricted to $S_c$. Define a homomorphic mapping $h = (h_S, h_I)$ from M to $M_c$ as follows:

$$h_S(s) = s_c \longleftrightarrow \bar{\delta}(s, \bar{x}) = \bar{\delta}(s_c, \bar{x}) \qquad (1)$$

for all $\bar{x} \in I^*$ with $\ell(\bar{x}) \geq n_1$, where $\bar{x} = \Lambda$ for all cyclic states, ie they map to themselves, and $n_1^*$ is the smallest integer for which equation (1) holds. Now, $h_S$ defines a congruence $R_{cy}(S)$ on S as follows:

$$s_1 \, R_{cy}(S) \, s_2 \longleftrightarrow h_S(s_1) = h_S(s_2).$$

Since no two cyclic states are related, and each block of $R_{cy}(S)$ contains one and only one such state, it is clear that this partition is computed by the cyclic submachine.

Thus, if $R_{cit}(S) \cdot R_{cy}(S) = 0$, then the parallel decomposition holds. An example of these partitions is given in Fig. 4.1 for machine $M_{4.1}$.

| | 0 | 1 |
|---|---|---|
| 1 | 1 | 8 |
| 2 | 1 | 8 |
| 3 | 4 | 5 |
| 4 | 4 | 5 |
| 5 | 4 | 5 |
| 6 | 4 | 5 |
| 7 | 1 | 8 |
| 8 | 1 | 8 |

$M_{4.1}$

$R_{ci}(S) = \overline{1,8;2;3;4,5;6;7}$

$R_{cit}(S) = \overline{1,4,5,8;2;3;6;7}$

$R_{cy}(S) = \overline{1,7;2,8;3,4;5,6}$

Fig. 4.1. Partition for Machine $M_{4.1}$.

* Note if $n_1$ does not exist for M, then h and $R_{cy}$ are not defined. (see 4.2)

The product of the two partitions $R_{cit}(S) \cdot R_{cy}(S) = 0$. The $R_{cy}(S)$ partition is unique for each homomorphic mapping from the machine to its submachine; but, for a given machine, the mapping itself may not be unique.

## 4.2. Non-Decomposable Logic-Free Machines

A machine is logic-free realizable if and only if it is an image of a machine which consists of the parallel combination of a definite machine with a CI machine. Certain properties of these machines and their images will now be defined, and then state splitting techniques based on these properties will be described. These techniques will generate a projection of any logic-free realizable machine for which there will exists a parallel decomposition on its state behaviour into a cyclic logic-free realizable machine and a CIT machine.

### Properties of Logic-Free Realizable Machines

Definition 4.1: The connected submachine of some state s which belongs to some machine $M = (S, I, \delta)$ is defined as $M_s = (S_s, I, \delta_s)$ where $S_s \subseteq S$ is the set of all states which can be reached from s, and $\delta_s$ is $\delta$ restricted to $S_s$.

Theorem 4.1: Every machine $M = (S, I, \delta)$ which consists of the parallel combination of a definite machine with a CI machine has the following properties:

P(1)- for every state $s \in S$ there exists some cyclic state s' and some integer n such that

$$\bar{\delta}(s, \bar{x}) = \bar{\delta}(s', \bar{x}), \quad \text{for all } \bar{x} \in I^* \text{ with } \ell(\bar{x}) \geq n.$$

P(2)- The connected submachine of every cyclic state is strongly connected. (This property guarantees that the set of cyclic states constitute a submachine).

Proof: Let $M_1 = (D, I, \delta_1)$ be the definite machine, and $M_2 = (A, I, \delta_2)$ be the CI machine; ie. $S = D \times A$.

P(1)- It is easily verified that P(1) holds for any definite or CI machine. Thus, for any $(a_i, d_j) \in S$ where $a_i \in A$ and $d_j \in D$, there exists some cyclic states $a_r, d_k$ and some integers $n_1, n_2$ such that

$$\bar{\delta}_2(a_i, \bar{x}) = \bar{\delta}(a_r, \bar{x}) \text{ for all } \bar{x} \text{ with } \ell(\bar{x}) \geq n_1$$

and $$\bar{\delta}_1(d_j, \bar{x}) = \bar{\delta}_1(d_k, \bar{x}) \text{ for all } \bar{x} \text{ with } \ell(\bar{x}) \geq n_2$$

and so $$\left(\bar{\delta}_2(a_i, \bar{x}), \bar{\delta}_1(d_j, \bar{x})\right) = \left(\bar{\delta}_2(a_r, \bar{x}), \bar{\delta}_1(d_k, \bar{x})\right) \text{ for all } \bar{x} \text{ with } \ell(\bar{x}) \geq \max(n_1, n_2)$$

$$= \bar{\delta}((a_i, d_j), \bar{x}) = \bar{\delta}((a_r, d_k), \bar{x}) \text{ for all } \bar{x} \text{ with } \ell(\bar{x}) \geq \max(n_1, n_2)$$

but $(a_r, d_k)$ is cyclic, so that P(1) holds.

P(2)- It is clear that the connected submachine of every cyclic state of a definite machine or a CI machine is strongly connected. Let p be the lowest common multiple of all cycles of $M_2$, and assume $M_1$ is k-definite (that is, any two states map to the same state under an input sequence of length greater than or equal to k). Let q be an integer such that $ap = q \geq k$ for some integer a. Then, any state $(a_i, d_k)$ will map to all states of the form $(a_i, D_c)$ under all input sequences of length $\geq q$, where $D_c$ stands for all states of the connected submachine of $d_k$. However, $a_i$ will reach all states $A_c$ of its connected submachine, so that all states in $A_c \times D_c$, can be reached, which is clearly the connected submachine of $(a_i, d_k)$. Thus, P(2) holds.

Examples of machines with and without these properties are shown in Fig. 4.2.

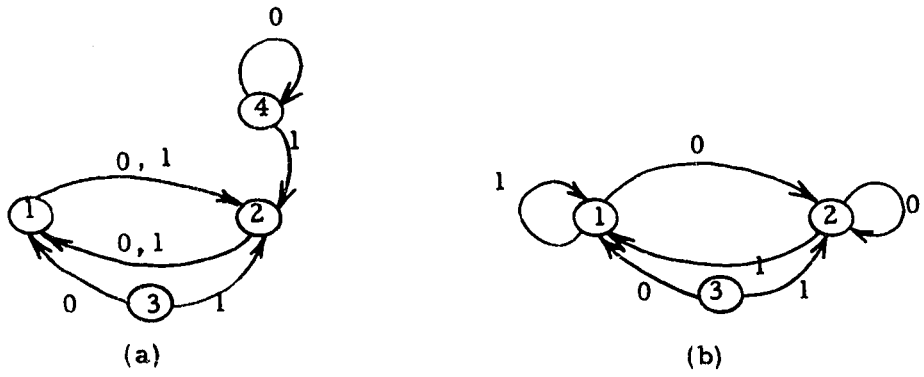



Fig. 4.2. (a) Flow Graph of a Machine without P(1) and P(2).
(b) Flow Graph of a Machine with P(1) and P(2).

In the machine of Fig. 4.2 (a), state 4 is cyclic, but its connected submachine is not strongly connected. Thus, P(2) does not exist. State 3 is non-cyclic, and a careful check will show that it is not related to any of the cyclic states by P(1).

In the machine of Fig. 4.2 (b), however, all cyclic states have connected submachines which are strongly connected, and the non-cyclic state, 3, is related to state 1 with $\ell(\bar{x}) = 2$.

Theorem 4.2: The two properties P(1) and P(2) of Theorem 4.1 are preserved under homomorphisms.

Proof: Assume there exists a homomorphic mapping $h = (h_S, h_I)$ from some machine $M = (S, I, \delta)$ which enjoys P(1) and P(2) to some machine $M' = (T, I', \delta')$.

P(1)- If, for every state $s_1 \in S$ there exists some cyclic state $s_2$ and some integer n such that

$$\bar{\delta}(s_1, \bar{x}) = \bar{\delta}(s_2, \bar{x}) \text{ for all } \bar{x} \in I^* \text{ with } \ell(\bar{x}) \geq n,$$

then $h_S(\bar{\delta}(s_1, \bar{x})) = h_S(\bar{\delta}(s_2, \bar{x}))$ for all $\bar{x} \in I^*$ with $\ell(\bar{x}) \geq n$.

Thus, $\bar{\delta}'(h_S(s_1), h_I(\bar{x})) = \bar{\delta}'(h_S(s_2), h_I(\bar{x}))$ under the same conditions.

Since h is onto, this holds for all $t \in T$, and therefore, M' enjoys P(1).

P(2)- We will show that this property holds for all such image machines M' by proving:

1) that there exists a homomorphic mapping $h_c = (h_{Sc}, h_I)$ from the cyclic submachine $M_c = (S_c, I, \delta_c)$ to $M'_c = (T_c, I', \delta'_c)$, the cyclic submachine of M', where $S_c$ and $T_c$ are the sets of cyclic states.

2) that P(2) is preserved under homomorphic mappings.

Proof: 1) Let R(S) be any congruence on S. Define a congruence $R(S_c)$ on $S_c$ as

$$s_c \; R(S_c) \; s'_c \longleftrightarrow s_c \; R(S) \; s'_c \quad \text{for all } s_c \in S_c$$

It is easily shown that $R(S_c)$ is a congruence relation. We claim that if $R(S_c)$ defines $h_{Sc}$, and $h_I: I \rightarrow I'$, then $h_c$ maps from the cyclic submachine of M to the cyclic submachine of M'. This we will show by proving that a state of M' is a cyclic state if and only if it is an image of a cyclic state of M under $h = (h_S, h_I)$, and thus under $h_c$.

a) If some state s is cyclic, then by definition there exists some input sequence $\bar{x} \in I.I^*$ such that $\bar{\delta}(s, \bar{x}) = s$. This means that $h_S(\bar{\delta}(s, \bar{x})) = h_S(s)$ and then that $\bar{\delta}'(h_S(s), h_I(\bar{x})) = h_S(s) =$ some $t \in T$. Thus, by definition, t must be a cyclic state.

b) Alternately, if t is some cyclic state $\in T_c$, then there must exist some input sequence $\bar{x}' \in I'.I'^*$ such that $\bar{\delta}'(t, \bar{x}') = t$, which implies that $h_S^{-1}(\bar{\delta}'(t, \bar{x}')) = h_S^{-1}(t)$ where $h_S^{-1}(t)$ stands for the set of all states of M that map to t.

Now, pick any state of $h_S^{-1}(t)$, say $s_1$. Then, $\bar{\delta}(s_1, \bar{x})$ (where $\bar{x} \in h^{-1}(\bar{x}')$) must map to some other state of the same set. Similarly, that state under the same input sequence must map to another of the set. Since the set is finite, one state must eventually repeat itself, and therefore is cyclic. Call this state $s_c$. By definition, $s_c$ maps to t under $h_S$. Thus, t must be the image of at least one cyclic state of M.

It is clear, then, that the cyclic states of M map to the set of cyclic states of M' with an onto homomorphic mapping $h_c = (h_{Sc}, h_I)$.

2) Now, assume that some state $t_c \in T$ is cyclic, and furthermore, that $\bar{\delta}(t_c, \bar{x}') = t'$ for some $\bar{x}' \in I'.I'^*$. Then, t is the image of some cyclic state, say $s_c$. Now, for some $\bar{x} \in h_I^{-1}(\bar{x}')$,

$$\bar{\delta}(s_c, \bar{x}) = s'$$

That is, t' is an image of s'. However, because of P(2), there must exist some $\bar{x}_i \in I.\ I^*$ such that

$$\bar{\delta}(s', \bar{x}) = s_c.$$

This implies that

$$h_S(\bar{\delta}(s', \bar{x})) = h_S(s_c)$$

and that

$$\bar{\delta}'(h_S(s'), h_I(\bar{x}_i)) = h_S(s_c).$$

Therefore,

$$\bar{\delta}'(t', h_I(\bar{x}_i)) = t_c.$$

But this means that the connected submachine of $t_c$ is strongly connected, and P(2) exists for M'.

Corollary 1: If a machine M' is logic-free realizable, it must enjoy properties P(1) and P(2).

These properties provide a preliminary test for logic-free realizability. If either P(1) or P(2) fail to hold for some machine M, then that machine cannot be realized on a logic-free circuit.

Corollary 2: If a machine M' is logic-free realizable, there exists a homomorphic mapping from the cyclic submachine of some sequential machine which consists of the parallel combination of a CI machine with a definite machine, to the cyclic submachine of M'.

Corollary 3: The cyclic submachine $M_c = (S_c, I, \delta_c)$ of some machine M is logic-free realizable if and only if the two partitions $R_{ci}(S_c)$ and $R_{ff}(S_c)$ defined over the states of the projection of $M_c$ generated by the input successor tree, multiply to 0. Further, this property is necessary if M itself is to be logic-free realizable.

Proof: The cyclic submachine of any machine which consists of the parallel combination of a definite machine with a CI machine itself consists of the parallel combination of a definite strongly connected machine and a CIP machine. (This is a special case of Theorem 2.6). The conditions, then, of this corollary parallel those of the cyclic machine in section 3.2. That this condition is necessary follows directly from the theorem.

This corollary suggests that the cyclic submachine of any logic-free realizable machine should be realized independently of the rest of the machine.

Before outlining the general algorithm leading to the logic-free realization of a sequential machine, we present the following graph called the pair graph, which will be essential in the eventual assignment.

## The Pair Graph

Given some machine which enjoys P(1) and P(2); select the set of primary states, and pair each up with one of the cyclic states it relates to by P(1). Order the pair so that the primary state appears first. These will be called 'primary pairs'. Now, for each primary pair construct a successor tree in the following way; 1) The first level of the tree consists of one primary pair; the second level contains all successor pairs of the first each ordered so that the successor of the primary state appears first. Transitions between pairs are indicated. The third level contains those successors of the second, ordered according to their predecessors, etc. 2) Each tree is terminated at a level in which each node contains a pair of identical cyclic states. The pair graph consists of the set of all such trees.

Now P(1) ensures that each tree of any logic-free realizable machine will terminate after a finite number of steps, and P(2) guarantees that the second state of each pair is cyclic.

## State Splitting on the Pair Graph

The first state of each pair that is not in the last level of some tree becomes a non-cyclic state if it is not one already. Thus, each first state is split as many times as it appears in all but the final levels.

One example of a pair graph is given in Fig. 4.3 for $M_{4,2}$. The projection generated by the pair graph is shown in Fig. 4.4.

The pair graph is a very powerful method for state splitting, and it guarantees that both partitions $R_{cit}(S)$ and $R_{cy}(S)$ hold for the projection it generates.

| | 0 | 1 |
|---|---|---|
| 1 | 5 | 1 |
| 2 | 4 | 2 |
| 3 | 5 | 1 |
| 4 | 6 | 2 |
| 5 | 5 | 3 |
| 6 | 6 | 2 |
| 7 | 6 | 2 |
| 8 | 7 | 2 |
| 9 | 7 | 8 |
| 10 | 6 | 2 |

$M_{4.2}$

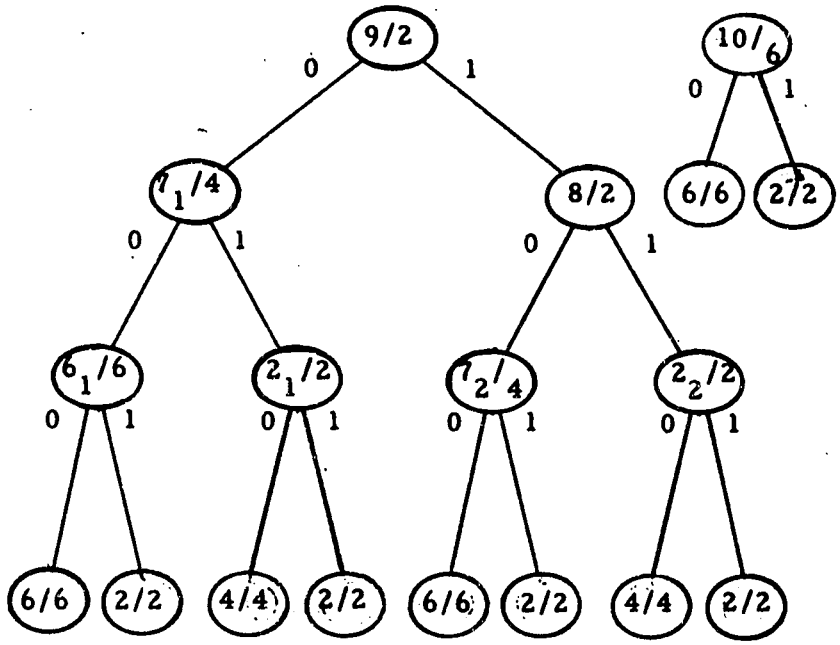



Fig. 4.3. Pair Graph and Flow Table for Machine $M_{4.2}$.

| | 0 | 1 |
|---|---|---|
| 1 | 5 | 1 |
| 2 | 4 | 2 |
| $2_1$ | 4 | 2 |
| $2_2$ | 4 | 2 |
| 3 | 5 | 1 |
| 4 | 6 | 2 |
| 5 | 5 | 3 |
| 6 | 6 | 2 |
| $6_1$ | 6 | 2 |
| $7_1$ | $6_1$ | $2_1$ |
| $7_2$ | 6 | 2 |
| 8 | $7_2$ | $2_2$ |
| 9 | $7_1$ | 8 |
| 10 | 6 | 2 |

Fig. 4.4. Flow Graph of the Projection of $M_{4.2}$ Generated by the Pair Graph.

The Partitions

$R_{cit}(P(M))$

Place all cyclic states into one block and no other states in that block. Place all first states of each pair in the same block if and only if they are in the same level of the same tree. It is clear that the resultant partition is computed by the largest CIT component machine of the projection. The partition for $P(M_{4.2})$ is $\overline{1,2,3,4,5,6};\overline{2_1,6_1,2_2,7_2};\overline{7_1,8};\overline{9};\overline{10}$

$R_{cy}(P(M))$

Place in each block one and only one cyclic state, and with it those non-cyclic states which pair up with it on the pair graph. Some reflection will show that such a partition will have S. P., and since each block contains one and only one state, that it is, in fact, computed by the cyclic submachine, $R_{cy}(P(M_{4.2})) = \overline{2,2_1,2_2,8,9};\overline{4,7_1,7_2};\overline{6,6_1,10};\overline{1};\overline{3};\overline{5}.$

There is one more requirement if the machine is to be logic-free realizable using previously devised methods. The product of the two partitions must equal the trivial (zero) partition. The state splitting procedure afforded us by the pair graph does not guarantee this. However, we will show that the state splitting method used to realize the cyclic submachine can guarantee that no two states in the same level of the tree, and thus in the same block of the CIT partition, will pair with the same cyclic state, and thus belong to the same block of the cyclic partition. Clearly, this guarantees that the two partitions will multiply to zero. This property is best illustrated by developing the general assignment procedure, and illustrating the process with an example. The example used will be $M_{4.2}$.

Development of the Assignment Procedure

Given a machine M; to determine if it is logic-free realizable;

1) generate the input successor tree over all the states of M. The set of states in the nodes of its final level belong to the cyclic submachine, $M_c = (S_c, I, \delta_c)$. (This is guaranteed by P(1)). Now, proceed to realize the cyclic submachine by applying the input successor tree to its states. Normally, this tree is terminated just before a set of nodes in any level appears twice. However, it is valid to extend the tree beyond this level, thus generating a larger projection circuit. Assume then, that the tree is stopped at some level g. The tree has the following property: each node of its final level is assigned a different code - precisely the reverse of the input sequence required to map to it. Clearly, this implies that no two states with a common predecessor under some input sequences $\bar{x}_1, \bar{x}_2 \in I.I^*$ with equal lengths less than or equal to g, will be assigned the same code. In the projection, then, they cannot be the same state. This property will prove very important later in the construction.
2) Continuing with the assignment; each primary state is paired up with one of the cyclic states it corresponds to under P(1). The pair graph is then generated.

The pair graph for $M_{4.2}$ is given in Fig. 4.3, and the input successor trees in Fig. 4.5. The projection of the cyclic submachine generated by the input successor tree is given in Fig. 4.6.

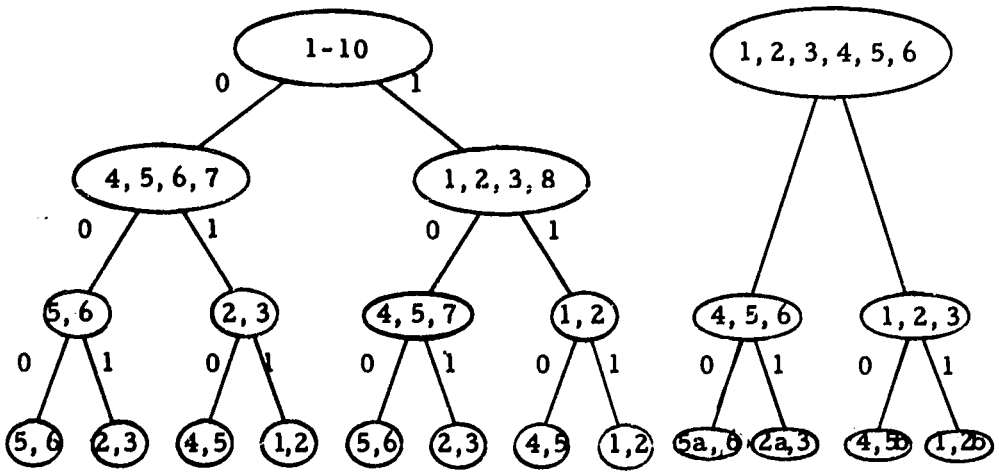



Fig. 4.5. Input Successor Trees for Machine $M_{4.1}$.

| | 0 | 1 |
|---|---|---|
| 1 | $5_b$ | 1 |
| $2_a$ | 4 | $2_b$ |
| $2_b$ | 4 | $2_b$ |
| 3 | $5_b$ | 1 |
| 4 | 6 | $2_a$ |
| $5_a$ | $5_a$ | 3 |
| $5_b$ | $5_a$ | 3 |
| 6 | 6 | $2_a$ |

Fig. 4.6. The Projection of the Submachine Generated by the Input Successor Tree.

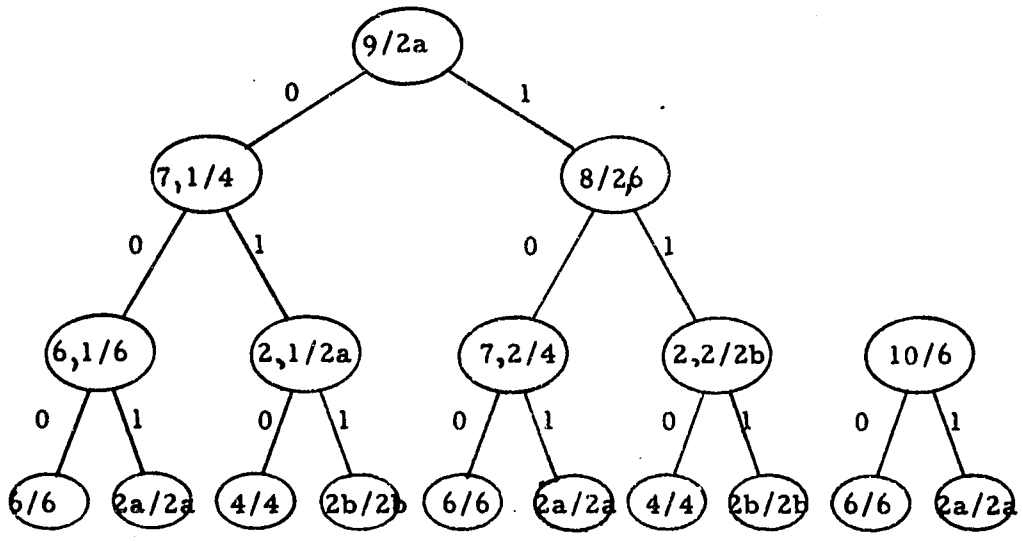



Fig. 4.7. Pair Graph for the Total Projection of $M_{4.1}$.

This done, we pose the following question: under what conditions will two non-cyclic states of the same level in the same tree pair up with the same cyclic state of the complete projection? There are two possible types of machines:

1) those for which the longest tree of the pair graph has length, $L \leq g + 1$. In this case, no two non-cyclic states of the same level of the same tree could pair up with the same cyclic state, for this would imply that those two cyclic states, which have a common cyclic predecessor (that state which pairs up with the primary state of the tree) but were mapped to under different input sequences of equal length less than g, were equal. This is a clear violation of the property of the assignment from the tree. The problem would not occur for this case, therefore. $M_{4.2}$ enjoys this condition.
2) Those for which $L > g + 1$. In this case it <u>is</u> possible that some non-cyclic states pair up with the same cyclic state

on the same level of the same tree. In this event, however, it is only necessary to extend the input successor tree applied to the cyclic submachine until g = L - 1, and so eliminate the condition.

3) the rest of the procedure is obvious. Generate the partitions $R_{cit}$ and $R_{cy}$ over the projection. Construct the tables for the machine which computes $R_{cy}$, and determine the smallest input independent partition and that partition computed by the largest feedback-free front machine. Test if they multiply to zero, and if they do, realize without logic each machine that computes them.

The final solution for $M_{4.2}$ is given in Figs. 4.8, 4.9 and 4.10.

| | 0 | 1 |
|---|---|---|
| 1 | $5_b$ | 1 |
| $2_a$ | 4 | $2_b$ |
| $2_b$ | 4 | $2_b$ |
| 3 | $5_b$ | 1 |
| 4 | 6 | $2_a$ |
| $5_a$ | $5_a$ | 3 |
| $5_b$ | $5_a$ | 3 |
| 6 | 6 | $2_a$ |
| $2_1$ | 4 | $2_b$ |
| $2_2$ | 4 | $2_b$ |
| $6_1$ | 6 | $2_a$ |
| $7_1$ | $6_1$ | $2_1$ |
| $7_2$ | 6 | $2_a$ |
| 8 | $7_2$ | $2_2$ |
| 9 | $7_1$ | 8 |
| 10 | 6 | $2_a$ |

Fig. 4.8. Flow Table for P(M) of Machine $M_{4.2}$.

The partitions for P(M) are:

$$R_{cit}(P(M)) = \overline{1, 2_a, 2_b, 3, 4, 5_a, 6};\ \overline{6_1, 2_1, 7_2, 2_2};\ \overline{7_1, 8};\ \overline{9};\ \overline{10}$$

$$R_{cy}(P(M)) = \overline{1};\ \overline{2_a, 2_1, 9};\ \overline{2_b, 2_2, 8};\ \overline{3};\ \overline{4, 7_1, 7_2};\ \overline{5_a};\ \overline{5_b};\ \overline{6, 6_1, 10}$$

Relabelling the blocks of $R_{cy}(P(M))$ as A;B;C;D;E;F;G;H: respectively, we construct the flow table for the cyclic machine,in Fig. 4.9.

| | 0 | 1 |
|---|---|---|
| A | G | A |
| B | E | C |
| C | E | C |
| D | G | A |
| E | H | B |
| F | F | D |
| G | F | D |
| H | H | B |

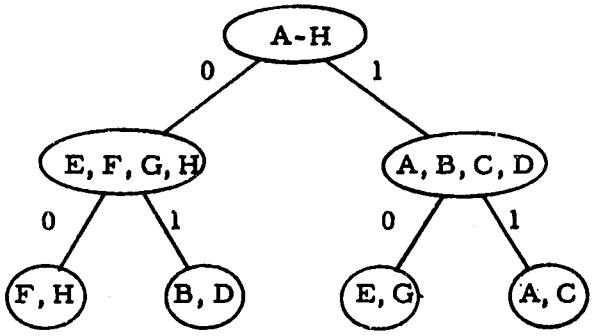



Fig. 4.9. Flow Table for the Cyclic Component Machine of $M_{4.2}$.

No state splitting is necessary; the partitions are:

$$R_{ci}(S) = \overline{A, D, F, G};\ \overline{B, C; E, H}$$

$$R_{ff}(S) = \overline{A, C; B, D; E, G; F, H}$$

| | Definite Strongly Connected | CIP | CIT |
|---|---|---|---|
| 1 | 11 | 0 | 0000 |
| $2_a$ | 10 | 1 | 0000 |
| $2_b$ | 11 | 1 | 0000 |
| 3 | 10 | 0 | 0000 |
| 4 | 01 | 1 | 0000 |
| $5_a$ | 00 | 0 | 0000 |
| $5_b$ | 01 | 0 | 0000 |
| 6 | 00 | 1 | 0000 |
| $2_1$ | 10 | 1 | 0010 |
| $2_2$ | 11 | 1 | 0010 |
| $6_1$ | 00 | 1 | 0010 |
| $7_1$ | 01 | 1 | 0100 |
| $7_2$ | 01 | 1 | 0010 |
| 8 | 11 | 1 | 0100 |
| 9 | 10 | 1 | 1000 |
| 10 | 00 | 1 | 0001 |

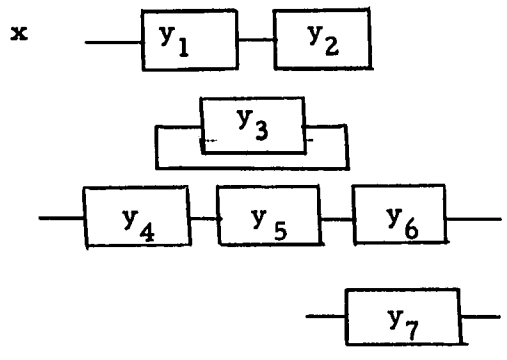
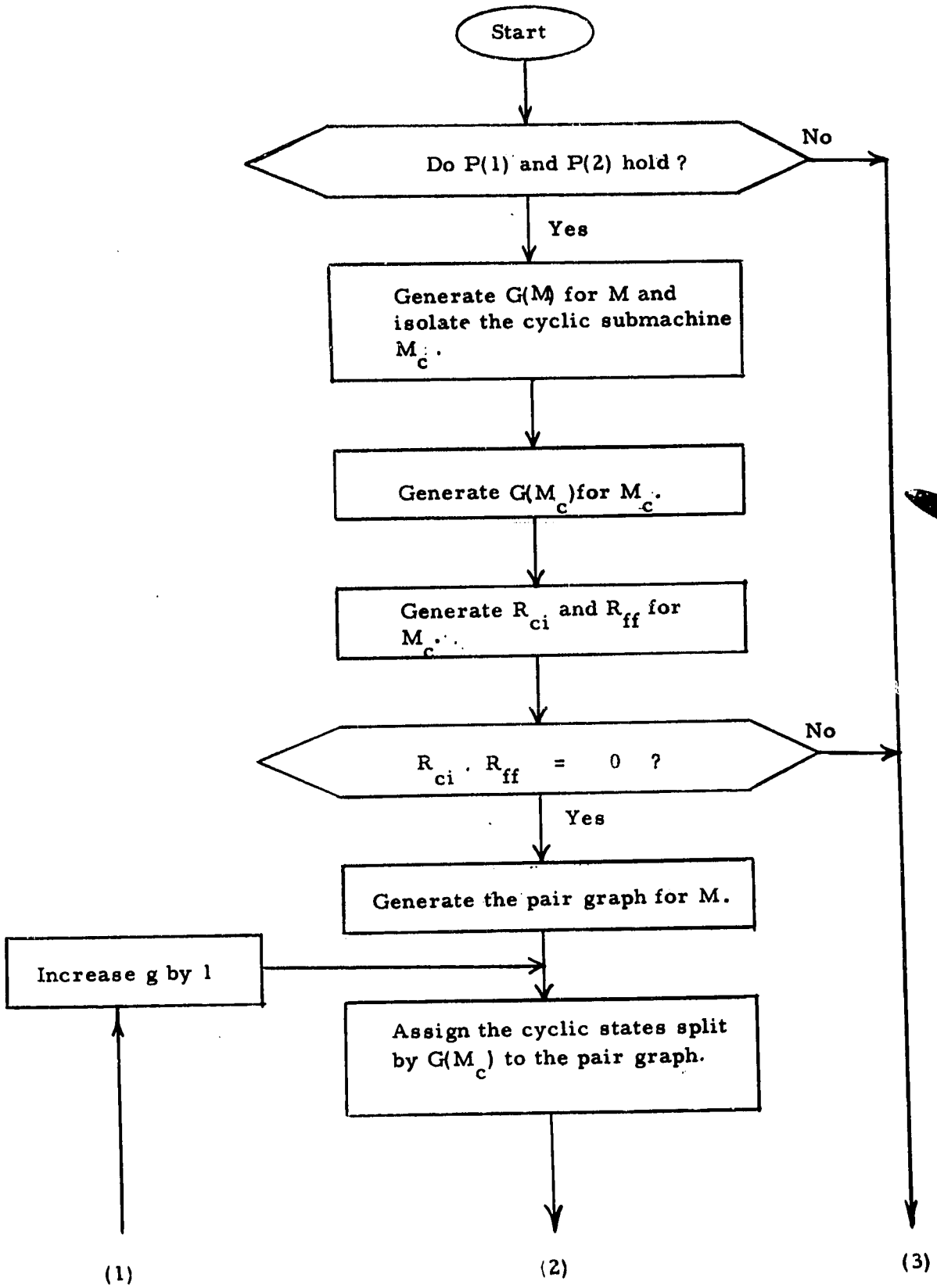



Fig. 4.10. Assignment and Circuit for $P(M_{4,2})$

The algorithm is presented in the Flow Table of Fig. 4.11.

The Algorithm.



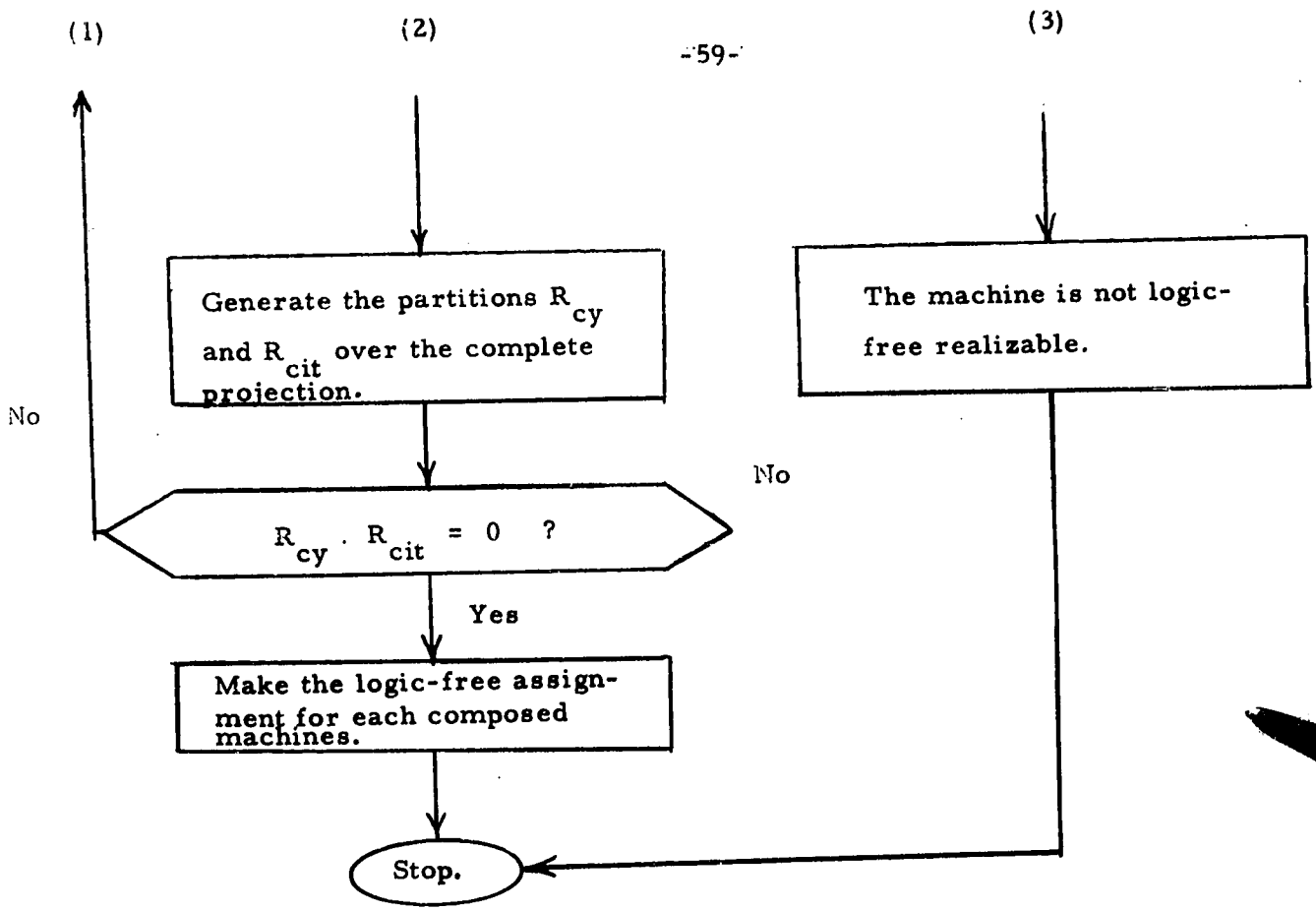



Fig. 4.11. Flow Table of the Algorithm.

## 4.3 Autonomous and Definite Machines

The preceding algorithm applies to machines in general, and therefore CI and definite machines in particular. Each of these machines has special characteristics brought out by this assignment approach.

### CI Machines

Some reflection will show that for every CI machine $M = (A, I, \delta)$, $R_{cy}(A) \cdot R_{cit}(A) = 0$ i.e. that it can be decomposed on its state behaviour into a CIP machine in parallel with a CIT machine. The algorithm greatly simplifies the assignment of such machines which have non-cyclic states, therefore.

### Definite Machines

From the definition of definite machines it is clear that any primary non-cyclic state can be related to any cyclic state by P(1). There is a large number of possible pair graphs for such machines, therefore.

In spite of this, the general algorithm is better defined and presents a simpler and often more economical logic-free realization for definite machines than the method described in section 2.

Examples are given in the following figures.

Example 1

| | 0 | 1 |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 1 | 2 |
| 3 | 1 | 2 |
| 4 | 2 | 1 |
| 5 | 3 | 2 |
| 6 | 3 | 4 |

Fig. 4.12. Flow Table for Machine $M_{4.3}$.

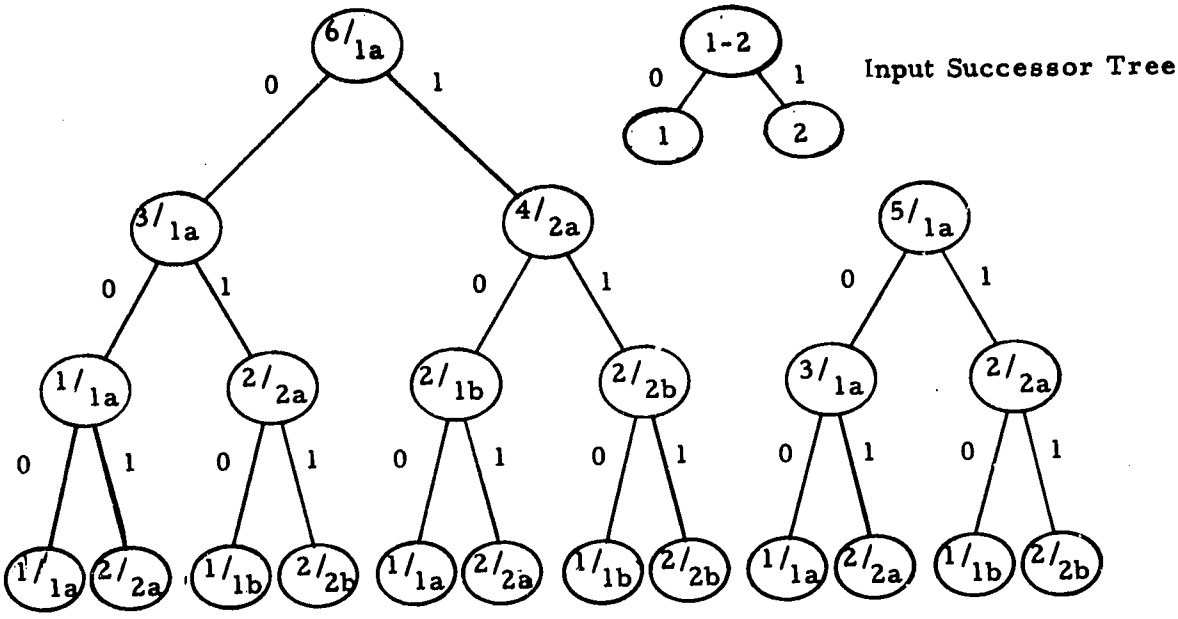



Fig. 4.13. Input Successor Tree and Pair Graph of Machine $M_{4.3}$.

It is evident that the tree will have to be extended one more level. This extension is shown in Fig. 4.14.

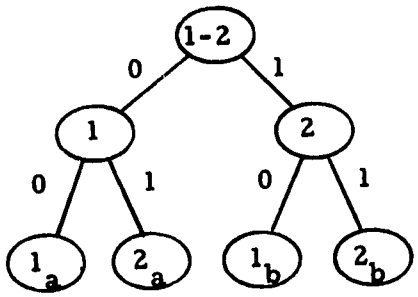



Fig. 4.14. Extension of the Input Successor Tree of Fig. 4.8.

The Flow Table for the Final Projection is given in Fig. 4.15.

| | 0 | 1 |
|---|---|---|
| $1_a$ | $1_a$ | $2_a$ |
| $1_b$ | $1_a$ | $2_a$ |
| $2_a$ | $1_b$ | $2_b$ |
| $2_b$ | $1_b$ | $2_b$ |
| $1_1$ | $1_a$ | $2_a$ |
| $1_2$ | $1_a$ | $2_a$ |
| $1_3$ | $1_a$ | $2_a$ |
| $1_4$ | $1_b$ | $2_b$ |
| $2_1$ | $1_2$ | $2_3$ |
| $2_2$ | $1_b$ | $2_b$ |
| $2_3$ | $1_b$ | $2_b$ |
| $2_4$ | $1_b$ | $2_b$ |
| $2_5$ | $1_a$ | $2_a$ |
| $3_1$ | $1_a$ | $2_a$ |
| $3_2$ | $1_a$ | $2_a$ |
| 4 | $1_b$ | $2_b$ |
| 5 | $1_a$ | $2_a$ |
| 6 | $1_a$ | $2_a$ |

Fig. 4.15. Flow Table of P(M) of Machine $M_{4,3}$.

The partitions are given by:

$$R_{cit}(P(M)) = \overline{1_a, 1_b, 2_a, 2_b}; \overline{1_3, 1_4, 2_4, 2_5}; \overline{1_1, 1_2, 2_2, 2_3}; \overline{3_2, 4}; \overline{2_1, 3_1}; \overline{5}; \overline{6}$$

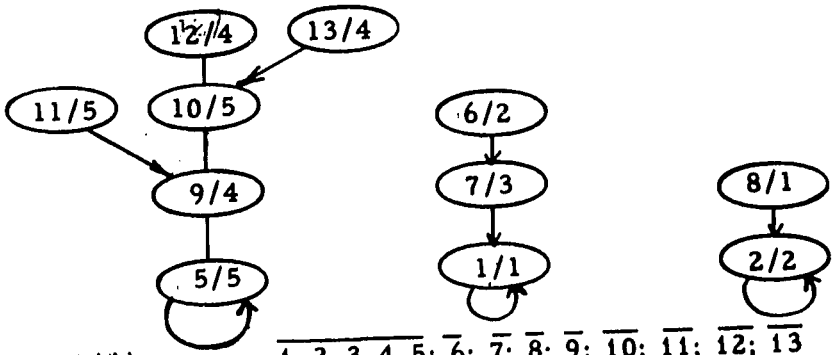
$$R_{cy}(P(M)) = \overline{1_a, 3_1, 1_1, 6, 3_2, 1_3}; \overline{2_a, 2_1, 2_2, 2_4}; \overline{1_b, 1_2, 2_5}; \overline{2_b, 2_3, 1_4}$$

Since the machine is definite, there is no need to generate any other partitions.

Example 2

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 1 |
| 4 | 5 |
| 5 | 4 |
| 6 | 7 |
| 7 | 1 |
| 8 | 2 |
| 9 | 5 |
| 10 | 9 |
| 11 | 9 |
| 12 | 10 |
| 13 | 10 |

Fig. 4.16. Flow Table of Machine $M_{4.4}$.




$R_{cit}(A) = \overline{1,2,3,4,5}; \overline{6}; \overline{7}; \overline{8}; \overline{9}; \overline{10}; \overline{11}; \overline{12}; \overline{13}$

$R_{cip}(A) = \overline{1,8}; \overline{2,6}; \overline{3,7}; \overline{4,9,12,13}; \overline{5,10,11}$

Fig. 4.17. Pair Graph of $M_{4.4}$, and Partitions.

## 5. CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER RESEARCH

That class of state machines which can be realized by logic-free circuits has been fully characterized. Methods of testing for such machines have been developed and assignment produces leading to logic-free realizations on parallel circuits, presented.

There are, however, more problems beyond the scope of this paper which could be considered. These are:

1) A characterization of machines with logic-free output functions as well as logic-free state realizations.

2) The logic-free realization of machines as a serial connection of a CI machine with a definite machine. Wayne Davis* has suggested that a necessary and sufficient condition for the logic-free realization of any machine is that the smallest front machine with $\Lambda^{\text{all the feedback.}}$ be CI. The sufficiency is easily shown, but its necessity does not follow immediately from the theory developed in this paper.

* personal communication

## REFERENCES

(1) J. Hartmanis, R.E. Stearns, "Algebraic Structure Theory of Sequential Machines," Prentice-Hall 1966.

(2) M.A. Harrison, "Introduction to Switching and Automata Theory," McGraw-Hill 1965.

(3) J.A. Brzozowski, "An Essay on Feedback," Technical Report No. 65-2, University of Ottawa 1965.

(4) M. Perles, M.O. Rabin, E. Shamir, "The Theory of Definite Automata," IEEE Trans. on Circuit Theory, Ec-12, (1965) 233-243.

(5) D.N. Arden, "Delayed Logic and Finite State Machines," Proc. 2nd Ann. Symp. on Switching Circuit Theory and Logical Design, Chicago Ill., (1961) 1-35.

(6) J.A. Brzozowski, "On Single Loop Realizations of Automata," IEEE Conference Record on Switching Theory and Logical Design, (1963), pp. 84-94.

(7) I. Toda, "The Tree Set of a Linear Machine," IEEE Trans. on Electronic Computers, Ec-14, (1966) 954-957.

(8) W.A. Davis, "On Shift Register Assignments for Sequential Machines," IEEE Conference Record on Switching Theory and Logical Design (1965) pp. 71-84.

(9) Nievergild, "Partially Ordered Classes of Finite Automata," IEEE Conference Record on Switching Circuits and Logical Design (1965) pp. 229-235.

(10) S. S. Yau, "Autonomous Clocks in Sequential Machines," IEEE Transactions on Electronic Computers, Ec-14 (June 1965) pp. 467-472.

(11) G. Birkoff and S. MacLane, "A Survey of Modern Algebra," Macmillan (1941).

(12) B. Elspas, "The Theory of Autonomous Linear Sequential Machines," IRE Trans. Vol. CT-6, pp. 45-60 (1959).

(13) A. Gill, "Introduction to the Theory of Finite State Machines," McGraw-Hill 1962.